Indice

Introduzione

La presente tesi ha come obiettivo principale la realizzazione un sistema di visione in grado di analizzare e studiare i movimenti oscillatori di un liquido contenuto in un recipiente durante una movimentazione. Il sistema realizzato si basa su tecniche di visione computazionale e prevede l'estrazione del profilo del liquido per calcolarne poi l'angolo di inclinazione. Attraverso queste informazioni ottenute sará poi possibile studiare la dinamica e un possibile controllo del liquido in presenza di oscillazioni o movimenti. Il tutto viene applicato su un manipolatore parallelo Adept Four in cui viene inserito un supporto a C come organo terminale. Nel campo della visione computazionale problemi di questo tipo vengono trattati solitamente con il fine della semplice misurazione di un livello di un liquido, per esempio nel riempire in modo corretto una bottiglia. Il tutto si puó realizzare anche tramite sensori ad ultrasuoni, tramite una corretta collocazione di questi infatti é possibile misurare il livello di un liquido e poi calcolarne eventualmente (tramite anche piú punti di misurazione) il suo andamento nel tempo. Il campo della visione computazionale non si limita comunque allo studio dei fluidi ma anche agli altri oggetti in generale. Nella videosorveglianza risulta utile poter identificare la presenza di persone/oggetti in movimento. Anche nel mondo industriale la visione aiuta spesso le attivitá umane nella rilevazione di oggetti indesiderati o nell'identificazione di eventi non previsti. L'esempio classico applicato alla robotica prevede la rilevazione anticipata di un ostacolo e la fase di superamento di questo senza collisione. Un altro caso classico riguarda l'ambito tessile in cui si controlla un filo nella sua integritá durante la lavorazione, cosa analoga con l'analisi dell'andamento di un liquido, in cui il filo rappresenta la superficie del fluido anche se, in questo caso, non si studia la sua integritá bensí il suo livello. A partire da questi presupposti l'argomento di tesi prevede quindi la realizzazione di un sistema per lo studio della movimentazione di un fluido, tralasciando i metodi tradizionali tramite sensori, applicando un metodo di calcolo tramite visione per tracciare l'andamento di oscillazione al fine di studiarne la dinamica e un possibile controllo. Uno studio interessante riguarda anche i modi di vibrare di un liquido, studio

visto spesso a livello didattico tramite associazioni con modelli equivalenti composti da pendolo. Questa caratteristica verrá trattata nell'elaborato e descritta nella parte finale al fine di capire quali modi di vibrare si riesce a percepire e analizzare con il sistema di visione implementato.

Capitolo 1

VISIONE COMPUTAZIONALE

Lo scopo di questo primo capitolo é quello di introdurre le basi della visione computazionale e le relative applicazioni in ambito industriale in associazione al progetto di tesi e ai suoi obiettivi.

1.1 Computer Vision

La Visione é il senso che consente all'essere umano di studiare il mondo 3D, di localizzare e riconoscere gli oggetti presenti in una scena e di percepire i rapidi mutamenti dell'ambiente. La Visione Artificiale (Visione Computazionale o Computer Vision (CV)) invece si definisce come la disciplina che studia i modelli e metodi per abilitare le macchine alla comprensione e interpretazione delle informazioni visuali presenti in immagini fisse bidimensionali o in sequenze video. Lo scopo principale della visione artificiale é quello di emulare la visione biologica non solo come l'acquisizione di una fotografia bidimensionale di un'area ma soprattutto come l'interpretazione del contenuto di questa riproducendo sui calcolatori elettronici il percorso cognitivo compiuto dall'essere umano nell'interpretazione della realtá che lo circonda attraverso le immagini che esso percepisce per mezzo degli occhi.

La Computer Vision é un campo in rapida crescita, sia grazie alla disponibilitá di telecamere sempre piu precise e meno costose, sia grazie alle elevate capacitá di calcolo dei moderni computer. I livelli raggiunti sono anche il frutto dell'intensa attivitá di ricerca degli ultimi decenni, che dai primi anni

Ottanta ad oggi ha portato ad una crescita esponenziale delle pubblicazioni scientifiche di settore. I campi applicativi della Computer Vision sono innumerevoli. La telecamera e infatti uno strumento non invasivo in grado di operare ad elevate distanze, se dotata dell'ottica opportuna. La versatilitá é legata alla grande mole di informazioni che le immagini trasportano, ed alla possibilitá di estrapolare solo quelle necessarie alla specifica applicazione. A differenza di un comune laser per misurazioni, infatti, le immagini immagazzinano dati provenienti da diverse direzioni, veicolati dai raggi di luce emessi o riflessi dai corpi. D'altro canto, la precisione dei risultati puó essere fortemente condizionata da fattori che vanno dalla risoluzione della telecamera alla configurazione degli algoritmi. La Computer Vision trova sempre piu spazio nell'automazione dei processi industriali, nel controllo di qualitá, in applicazioni militari ed aerospaziali, nell'ingegneria edile e nell'architettura e nella sorveglianza. L'auto-localizzazione e la ricostruzione ambientale sono le tematiche della Computer Vision che presentano maggiori analogie con le funzioni del nostro apparato visivo. Data una sequenza di fotogrammi di uno stesso ambiente, ricavati con continuitá da una macchina fotografica o da una telecamera in movimento, l'algoritmo di auto-localizzazione riproduce la traiettoria del dispositivo nelle coordinate di un riferimento arbitrario. La ricostruzione ambientale prevede la generazione di un modello tridimensionale dell'ambiente, piu o meno dettagliato anche in base ad esigenze di real-time.

Il principio di funzionamento del sistema si basa, come la stereopsi nel mondo animale, sulla possibilitá di fondere in una mappa di profonditá le informazioni provenienti da due viste separate di una stessa scena.

La configurazione standard di un modello di Visione Computazionale viene rappresentata con una o piú telecamere connesse ad un computer che deve automaticamente interpretare le immagini di una scena reale, ottenendo informazioni utili per la navigazione, manipolazione ed il riconoscimento.

I sistemi di visione artificiale si articolano generalmente su tre livelli di astrazione:

1. **Basso Livello**
2. **Medio Livello**
3. **Alto Livello**

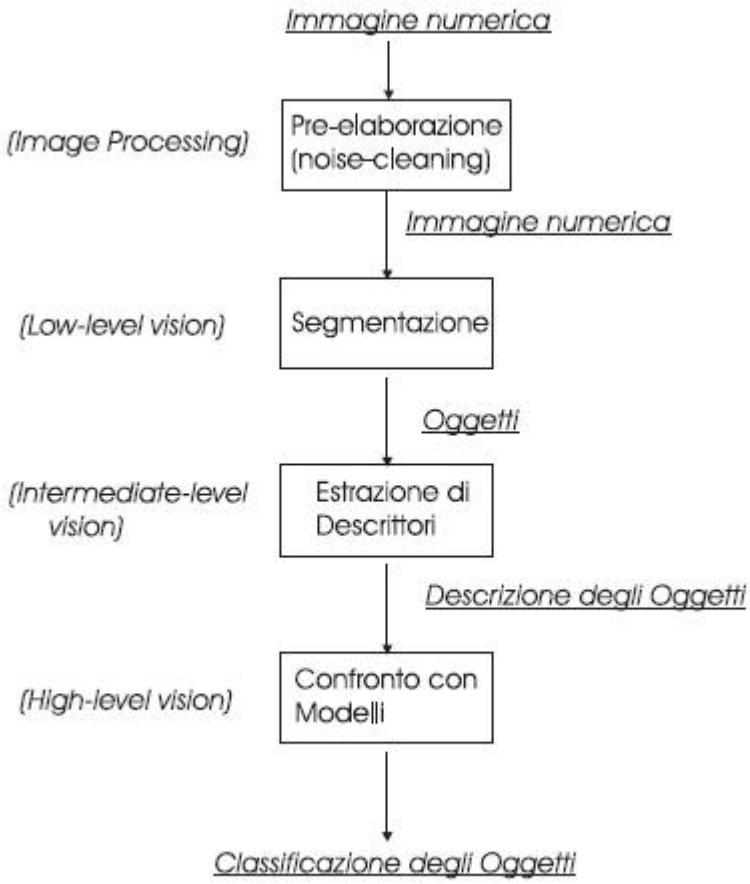


Figura 1.1: Astrazione Computer Vision

- La visione a basso livello (low-level-vision), data un'immagine in ingresso, ne viene prodotta una nuova (Image Reconstruction). Esalta alcune caratteristiche visuali in particolare l'estrazione di primitive geometriche, forma, profonditá, dimensione, contorni degli oggetti. Inoltre tenta di recuperare le degradazioni dovute, ad esempio, al rumore (Image Restoration). Consta di un gran numero di elaborazioni mirate a modificare l'immagine per evidenziarne determinati particolari. Tali elaborazioni (image processing) generalmente non producono direttamente nuova informazione, ma sfruttano solo l'informazione ricavabile dall'immagine stessa. Ovviamente tale fase deve essere affrontata con un certo criterio; infatti ogni scelta in questa fase si ripercuote pesantemente su ogni ulteriore passo di elaborazione successivo in quanto ogni algoritmo di piú alto livello ne utilizza i risultati.

- La visione a medio livello (mid-level-vision) si occupa dell'estrazione delle informazioni di tipo strutturale dall'immagine prodotta a livello inferiore; opera su immagini per estrarne informazioni di tipo strutturale. L'informazione strutturale riguarda la composizione dell'immagine, il numero e le relazioni spaziali (espresse in pixel) fra gli oggetti in essa presenti. Talvolta fa uso di informazione a priori. In questo livello dunque si ottengono informazioni sulla struttura dell'immagine e spesso sulla forma degli oggetti in essa trovati (un esempio ne é la segmentazione).

- La visione ad alto livello (High-level-vision) opera sulle informazioni provenienti dalla visione a medio livello per comporre un modello - semantico della scena. Lo scopo di questo livello é quello di arrivare ad una forma di comprensione della scena osservata, come puó essere il riconoscimento di oggetti e di relazioni spaziali tra gli oggetti. Analizza la scena da un punto di vista semantico.

1.2 Applicazioni Computer Vision

La visione artificiale sta sempre piú assumendo il ruolo di supporto nell'analisi di immagini e filmati. La quantitá di informazione contenuta in un'immagine, e ancor di piú in un filmato (sequenza di immagini), fa si che il compito di analizzarle possa, in molti contesti, risultare pesante per l'uomo, per la noiosa attesa di eventi rari o per l'elevato sforzo di concentrazione richiesto dal verificarsi di una rapida successione di eventi. Nelle applicazioni industriali per il controllo della qualitá, nella raccolta di dati per sistemi automatici o nell'analisi di flussi di immagini da telecamere adibite a videosorveglianza, ed in molti altri campi, la visione artificiale consente di limitare (se non eliminare) la necessitá di un controllo umano delle informazioni raccolte. Il grado di supporto dipende spesso in gran parte da quanto il tipo di ambiente da cui provengono le immagini possa ritenersi controllato e cioé dalla misura in cui le variazioni delle condizioni di ripresa possano discostarsi dal modello conosciuto dal sistema di visione che lo analizza. Data, in certi casi, l'impossibilitá di operare in condizioni controllate, l'obiettivo principale posto a chi opera ricerca in questo campo é di individuare algoritmi e sistemi il piú possibile immuni alle mutazioni rispetto al modello. Tale variabilitá é maggiore per immagini provenienti da ambienti all'aperto, mentre é piú contenuta per quelle provenienti da ambienti chiusi, come puó essere il caso di sistemi di ispezione industriale dei quali, ad esempio, esistono esemplari installati su linee di produzione seriale in grado di fornire ottimi risultati.

1.3 Immagini

Un'immagine é una funzione che rappresenta la misurazione di alcune caratteristiche (intensitá luminosa, colore) di una scena: é intrinsecamente bidimensionale mentre la scena risulta 3D. Un'immagine é di norma rappresentata come una matrice di elementi discreti, detti pixel, ciascuno di un colore.

Viene dunque definita come una funzione F(x,y) dell'intensitá luminosa il cui valore o ampiezza, corrispondente ad una generica coppia di coordinate spaziali (x,y), indica il valore dell'intensitá luminosa nel punto corrispondente. Nel caso specifico della Computer Vision la F(x,y) rappresenta la misurazione di alcune caratteristiche della scena reale tridimensionale, come ad esempio intensitá luminosa e colore. Le immagini sono acquisite utilizzando dispositivi che per la loro natura stessa introducono deformazioni, errori di quantizzazione ed in generale, rumore che devono essere opportunamen-

te filtrati o corretti. La funzione F(x,y) non é necessariamente scalare. Si consideri ad esempio il caso delle immagini a colori. é noto infatti, che ogni colore puó essere rappresentato come combinazione di una serie di colori fondamentali.

Lo spazio RGB viene rappresentato come un cubo, in cui lo spigolo in basso a sinistra rappresenta il nero, quello opposto il bianco, i grigi si trovano sul segmento che unisce il nero ed il bianco ed un generico colore sará rappresentato da un punto all'interno del cubo. L'occhio umano percepisce la somma delle 3 componenti come un unico colore.

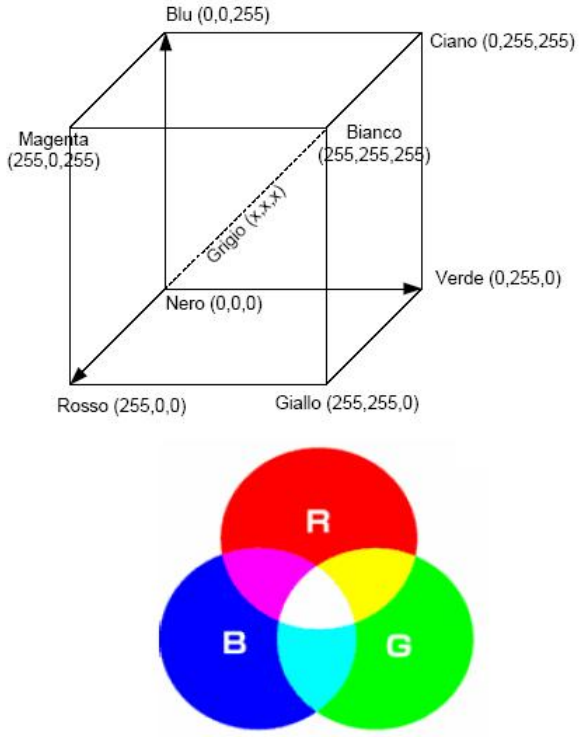


Figura 1.2: Spazio RGB

Pertanto si comprende facilmente che una qualsiasi immagine a colori puó essere rappresentata da una funzione vettoriale come segue: F(x,y)=[f1(x,y), f2(x,y), f3(x,y)] in cui ciascuna componente é detta canale ed indica la funzione di luminositá in funzione delle coordinate spaziali di uno dei colori fondamentali (si veda la rappresentazione dello spazio RGB in figura sopra). Se si considera anche il fatto che le immagini vengono percepite per mezzo della

luce riflessa dagli oggetti, é possibile considerare la F(x,y) come costituita da due componenti principali:

- Componente di illuminazione, indicata con i(x,y), che rappresenta la quantitá di luce diretta incidente sulla scena
- Componente di riflessione, indicata con r(x,y), che rappresenta la quantitá di luce riflessa dagli oggetti presenti sulla scena.

Allora un'immagine puó essere rappresentata in maniera alternativa come segue:

$$F(x, y) = i(x, y)r(x, y)$$

dove le componenti di illuminazione e riflessione risultano limitate secondo le relazioni:

$$0 \leq i(x, y) \leq \infty$$

$$0 \leq r(x, y) \leq 1$$

in cui per 0 si intende l'assorbimento totale mentre per 1 la totale riflessione.

Nel caso in cui si considerano immagini in movimento, se si vuole catturare la dinamica della scena, si deve introdurre nel modello una terza dimensione: la variabile tempo. Ne consegue che la funzione che descrive una sequenza di immagini a colori assumerá la forma seguente:

$$F(x, y) = [f1(x, y, t), f2(x, y, t), f3(x, y, t)]$$

Oltre alla rappresentazione RGB (forma additiva dei colori) ne esiste un altra in forma sottrattiva. Tale spazio di colori si basa sui tre colori, detti colori secondari, situati nel cubo RGB in corrispondenza degli spigoli opposti a quelli dei colori primari. Lo spazio é detto CMY (Cyan, Magenta, Yellow) e viene definito:

$$C = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \quad M = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \quad Y = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

mentra la trasformazione fra i due spazi di colore:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Lo spazio HSB (Hue, Saturation, Brightness) invece fornisce la rappresentazione del colore piú vicina al nostro modo di percepire i colori. La componente Hue rappresenta il colore dominante percepito dall'osservatore, ed é quindi legata alla lunghezza d'onda dominante presente nella luce percepita. La componente Saturation rappresenta la purezza del colore percepito, ed é tanto minore quanto piú il colore é diluito nella luce bianca, o se si preferisce é tanto maggiore quanto piú la lunghezza d'onda associata alla Hue é dominante. Ad esempio la Hue rosso puó avere diversi livelli di saturazione dal rosso completamente saturo, al rosa fino al bianco (nessuna saturazione). La componente Brightness rappresenta l'intensitá percepita.

La figura seguente mostra lo spazio HSB: la circonferenza rappresenta l'asse della Hue, su cui si trovano tutti i colori puri dello spettro. La distanza fra la circonferenza ed il centro rappresenta la Saturation. La Brightness vien rappresentata dall'asse verticale.

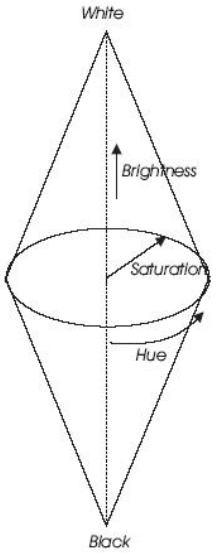


Figura 1.3: Spazio HSB

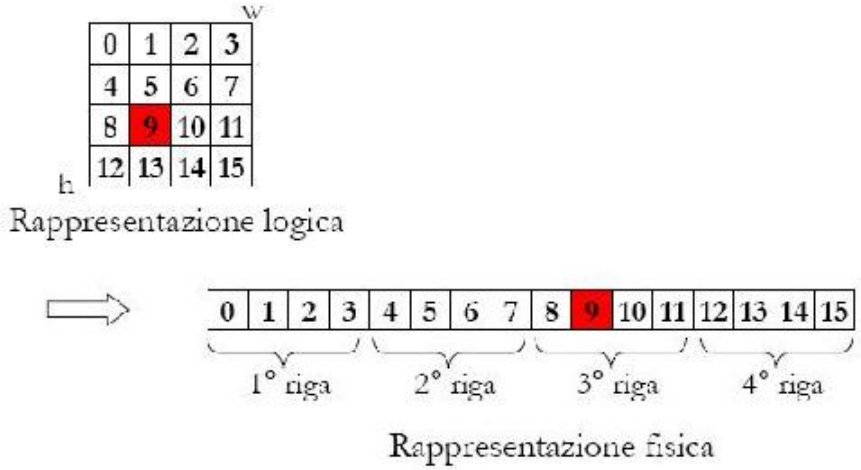
Si puó quindi osservare che H é data da un angolo, mentre S e B sono delle ampiezze. Inoltre, l'asse passante per il centro della circonferenza rappresenta i livelli di grigio.

1.3.1 Classificazione Immagini

I principali tipi di immagine sono i seguenti:

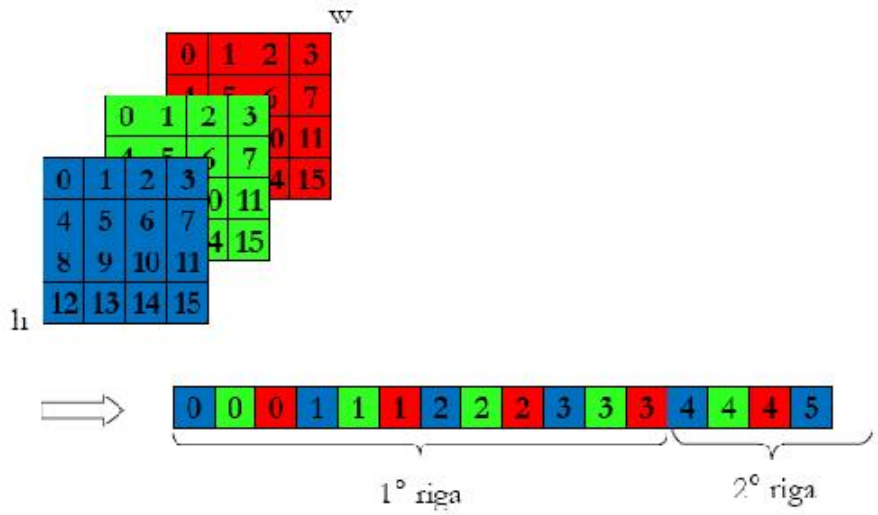
1. **Immagini a toni di grigio**
2. **Immagini a colori**
3. **Immagini binarie o in bianco e nero (B/W)**

Un'immagine a toni di grigio é rappresentata come una matrice di interi indicanti i valori di intensitá luminosa di ogni pixel su una scala di grigi. E' convenzione assumere i livelli discreti di grigio egualmente spaziati in un intervallo ben definito di valori, che normalmente va da 0 a 255. In genere il valore zero indica il nero, mentre il valore massimo indica il bianco. I valori compresi tra zero ed il valore massimo individuano tutti i toni di grigio intermedi. Questo tipo di immagine viene ampiamente utilizzato nella Computer Vision in quanto necessita di minore onerositá computazionale vista la trattazione di semplici matrici unidimensionali.



Per quanto riguarda le immagini a colori queste risultano piú complesse da

trattare vista la rappresentazione sia logica che fisica che hanno in memoria. Un colore infatti puó allora essere decomposto nella somma di tre colori fondamentali Rosso, Verde e Blu (RGB), ciascuno ovviamente preso con un'intensitá opportuna. Dato che lo spazio RGB é cartesiano, il colore di un pixel é rappresentato da un vettore e le sue componenti rappresentano la quantitá di rosso, verde e blu rispettivamente necessarie a ottenere quel colore (vedere immagini e trattazione sopra). Quindi un'immagine a colori puó essere interpretata tramite la una terna di matrici, ciascuna delle quali rappresenta un canale di colore: l'immagine nella sua interezza sará allora la sovrapposizione delle immagini corrispondenti a ciascuna matrice ed equivalentemente ogni canale preso da solo é una immagine a toni di grigio e quindi puó essere trattata di conseguenza.



Le immagini digitali binarie ricoprono un ruolo di particolare importanza nelle applicazioni di Computer Vision in quanto alla base dei processi di rilevamento e riconoscimento di oggetti. Di seguito verranno dunque illustrate le principali caratteristiche di questo tipo di immagini. Un'immagine é detta binaria se i suoi punti assumono solo due valori (tipicamente 0 e 1), ovvero se é costituita da soli due livelli di colore: bianco e nero. Si noti che si utilizza il termine punto indifferentemente da pixel in quanto le immagini digitali sono discretizzate, e quindi i punti sono rappresentati dai pixel; inoltre i pixel hanno forma rettangolare (come il monitor), ma di seguito verrá considerata, per semplicitá, come quadrata.

Generalmente si fissa l'origine del sistema di riferimento in alto a sinistra dello schermo, con gli assi orientati come nella figura seguente:

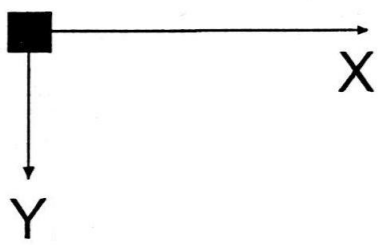


Figura 1.4: Sistema di riferimento utilizzato per le immagini

La scelta di questo particolare sistema di riferimento risulta valido per tutti i tipi di immagini.

1.4 OpenCV

La visione, intesa come pura acquisizione di immagini, é attualmente considerabile un problema giá risolto, o almeno un punto giá segnato, visto che le capacitá visive di sistemi ottici e relativi sensori hanno ampiamente superato le possibilitá dell'occhio umano in quanto a sensibilitá, velocitá e risoluzione. Il passo successivo, cioé la capacitá di interpretare ed utilizzare correttamente le informazioni acquisite, presenta invece ancora molti problemi insoluti. Convertire un'immagine in informazioni oggettive astraendone il contenuto dalla pura rappresentazione luminosa, sebbene sia un'operazione banale per un cervello umano adulto é, a tutt'oggi, un problema di elevata complessitá per un sistema automatico. Oltretutto Il campo di ricerca é evidentemente molto giovane, con meno di trent'anni di esperienza. In quest'ottica si inserisce la necessitá di una base comune di strumenti analitici, primo dei quali una libreria che raccolga le funzionalitá degli algoritmi piú utilizzati, oltre che una serie di formati di rappresentazione dei dati secondo standard aperti e condivisi. Le librerie OpenCV (Open Computer Vision) nascono e vengono utilizzate appunto a questo scopo. La Open Source Computer Vision Library é una libreria scritta in C e C++, costituita da oltre 500 funzioni utili nel campo dell'image processing e della computer vision. Punti di forza della libreria sono la completezza e il suo essere open source. Il fatto di essere liberamente distribuita garantisce sicurezza al codice e la possibilitá di apportare modifiche al codice stesso, assicurandone cosi una continua evoluzione. La licenza di distribuzione é priva di royalty e consente il suo utilizzo

anche in prodotti commerciali a condizione di mantenere le note di copyright. OpenCV costituisce quindi una infrastruttura aperta e gratuita, compatibile con la Intel Image Processing Library (IPL). Quest'ultima é una libreria non open source che esegue solo operazioni di image processinge dalla quale OpenCV ha ereditato originariamente alcune strutture dati. La portabilitá della libreria OpenCV é completa, sono infatti disponibili le versioni per i sistemi MS-Windows, Linux, BSD, Unix e MacOSX.

Lo sviluppo prende quindi le mosse da un gruppo di ricerca sponsorizzato da Intel. E' infatti parzialmente basata sulla Intel Image Processing Library (IPL). Tale prodotto é oggi integrato nella libreria commerciale IIPP (Intel Integrated Performance Primitives), con cui conserva piena compatibilitá, e che puó eventualmente rendere disponibili un completo ventaglio di funzioni di trattamento segnali (audio, video, sintesi vocale, crittografia, ecc) oltre che una migliore ottimizzazione delle prestazioni. Si avvale inoltre di numerosi contributi dalle piú svariate provenienze. A tal proposito l'elenco di credits citati dalla pagina ufficiale é quantomeno impressionante, si va da ricercatori del MIT, fino a docenti della Berkley University. Inutile sottolineare come questo offra giá una certa garanzia relativamente alla buona qualitá del codice e degli algoritmi applicati. Tra i punti di forza si sottolinea inoltre la politica di licenza utilizzata, in stile BSD. A grandi linee questo permette una libera ridistribuzione sia in forma sorgente che binaria, anche all'interno di prodotti commerciali, a condizione di mantenere, come giá detto, le note di copyright e di non utilizzare il nome Intel a scopo promozionale di prodotti derivati.

In una libreria grafica si identificano genericamente almeno tre famiglie di librerie i cui scopi sono sostanzialmente differenti:

1. I Toolkit, ovvero librerie di primitive per la creazione di oggetti grafici di interfaccia (finestre, icone, bottoni,ecc)

2. Librerie di rendering e multimedia, come DirectX e OpenGL, orientate alla massima performance nella creazione di effetti poligonali o vettoriali. L'utilizzo piú comune é teso all'ottenimento di elevate prestazioni grafiche sfruttate ad esempio nei videogiochi o nelle applicazioni multimediali.

3. Librerie di gestione hardware grafico, come digitalizzatori e framegrabber. Pur includendo tipicamente una base di funzioni di trattamento sono generalmente da considerarsi come API dei relativi driver hardware.

Le OpenCV, pur includendo alcune funzionalitá tipiche di ciascuna delle famiglie citate, non fanno parte di nessuno di questi gruppi. L'utilizzo primario é infatti quello collegato alla visione artificiale, il cui problema principale é quello di estrarre dalle immagini dati significativi e trattabili in modo automatico. Tale campo di studio trova le sue applicazioni piú comuni nella robotica, nei sistemi di videosorveglianza evoluti e nei sistemi di monitoraggio e sicurezza, oltre che in ogni sistema di archiviazione automatica di informazioni visive. La libreria include attualmente piú di 300 funzioni, che coprono le piú svariate esigenze di trattamento di immagini, comprese funzioni matematiche ottimizzate (elevamento a potenza, logaritmi, conversioni cartesiane-polari, ecc.) ed un completo pacchetto di algebra matriciale, sviluppato funzionalmente al resto del sistema.

Viene ribadita quindi la grande portabilitá, essendo disponibili versioni per tutte le varianti di Ms-Windows e per tutti i sistemi POSIX (Linux / BSD / UNIX / MacOsX), anche se il supporto ufficiale é, per ora, limitato ai primi. A questo proposito non ci si faccia spaventare dai numeri di versione (giunta alla 2.4) e dalla etichetta beta, in quanto il livello di stabilitá attualmente offerto assicura la possibilitá di utilizzo anche all'interno di ambienti di produzione. Il classico pacchetto installativo include, oltre ai binari ed agli header necessari alla compilazione, anche una discreta varietá di esempi, molto utili ad apprendere la sintassi e le tecniche di base. Sono inoltre disponibili diversi documenti introduttivi, oltre ad una reference in formato HTML. Per esigenze di spazio non ci si dilunga tanto sulle impostazioni di base necessarie ai differenti ambienti di sviluppo. Si sottolinea comunque come il problema della compatibilitá sia stato affrontato e risolto in modo particolarmente capillare, e quindi, tramite poche e semplici operazioni sará possibile essere subito produttivi all'interno dei piú comuni ambienti di sviluppo disponibili nel mondo Windows (VisualC++, Borland C++Builder, l'ottimo DevC++, ecc.), Linux (gcc, Kdevelop, Eclipse, ecc.) e Unix in generale.
La libreria é divisa in alcuni binari distinti (dll o so a seconda dei sistemi), motivo per cui il linking completo non é sempre necessario:

1. CxCore. E' l'oggetto principale nonché l'unico strettamente indispensabile. Include infatti tutte le funzioni di inizializzazione delle strutture utilizzate, l'algebra lineare e le altre funzioni di base.

2. Cv e CvAux. Includono praticamente tutte le funzioni di trattamento ed analisi, quindi il cuore delle OpenCv.

3. HighGui. Include alcune comode funzioni GUI, come ad esempio la tipica finestra di display, oltre alle funzioni di salvataggio e caricamento

immagini da file.

4. CvCam. Include funzioni di acquisizione video e di gestione delle telecamere (per il momento ancora piuttosto limitate).

1.4.1 cvBlob

OpenCv prevede nelle tante librerie a disposizione una raccolta di metodi per la trattazione esclusiva delle immagini binare e dell'estrazione delle informazioni da queste. Un immagine binaria, come giá visto, é composta da pixel con valori nulli o pari ad 1, solitamente associati rispettivamente, all'oggetto di interesse e allo sfondo. Risulta spesso utile andare ad analizzare e identificare questi oggetti rimasti dopo una fase di sogliatura per rendere l'immagine binaria, al fine di dedurre le loro informazioni. Attraverso la libreria cvBlob é possibile identificare gli oggetti binari nelle immagini in modo molto semplice e chiaro. Il metodo prevede l'identificazione dell'oggetto (chiamato blob) attraverso lo studio dei pixel che compongono l'immagine considerando la figura da estrarre come l'insieme dei pixel adiacenti con tutte le stesse caratteristiche (piú precisamente lo stesso valore, nullo nel caso delle immagini binarie). Dopo l'identificazione dell'oggetto é possibile con i metodi a disposizione, calcolare l'area dell'oggetto, il suo perimetro, il suo centro di massa, etcetc e molte altre informazioni di natura geometrica utili allo studio di questo. Gran parte del sistema di visione si basa su questi metodi soprattutto per il calcolo dell'orientazione di una figura di interesse.

1.5 Operazioni su immagini

In questo paragrafo e nei successivi saranno trattate alcune delle tecniche di image processing basate su operazioni spaziali, ossia operazioni eseguibili direttamente sui pixel di un'immagine. Le operazioni spaziali sono classificabili come operazioni puntuali quando eseguite pixel per pixel, operazioni locali quando trasformano il valore di un pixel in base ai valori dei pixel in un suo intorno, e in operazioni globali quando trasformano il valore di un pixel sulla base dei valori assunti da tutti i pixel dell'immagine.

Gli operatori logici AND, OR e NOT sono funzionalmente completi, in quanto ogni altro operatore puó essere definito mediante questi tre. Vengono usati ad esempio nelle operazioni morfologiche, nell'operazione di masking, e in quella di inversione di un'immagine. Si tratta in ogni caso di operazioni effettuate bit per bit tra pixel omologhi. Disponendo di due immagini composte

entrambe da una regione di pixel di foreground, il risultato dell'operazione OR consiste nell'insieme dei pixel appartenenti a una o all'altra regione, oppure ad entrambe, mentre l'operazione AND corrisponde all'insieme dei pixel in comune tra le due regioni. L'operazione NOT individua invece tutti i pixel che non sono nella regione.

1.5.1 Thresholding

La trasformazione di intensitá:

$$s = T(r) = \begin{cases} 0 & \text{se } r < m \\ 255 & \textit{altrimenti} \end{cases}$$

con $m \in [0, 255]$, riduce un'immagine a toni di grigio in un'immagine binaria.

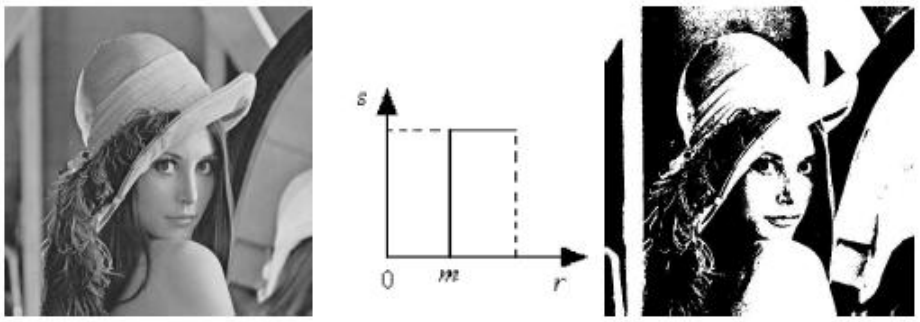


Figura 1.5: Esempio di sogliatura

Adottando una soglia unica per l'intera immagine la trasformazione denota come global thresholding. Quando la soglia varia in funzione di una qualche proprietá la trasformazione é nota invece come variable thresholding. In entrambi i casi queste operazioni permettono di separare gli oggetti o le regioni di interesse dallo sfondo dell'immagine. Nelle elaborazioni puntuali assume notevole importanza la conoscenza dell'istogramma, il quale consente di analizzare l'immagine fornendo una serie di informazioni statistiche utili in operazioni come la segmentazione. L'istogramma di un'immagine a toni di grigio indica il numero di pixel per ciscun livello di grigio. É una funzione

discreta $h(r_k) = n_k$, dove r_k é il k-esimo valore di intensitá ed n_k é il numero di pixel con intensitá r_k. E' in pratica una stima dell'occorrenza dei livelli di grigio, utile perché fornisce una descrizione globale dell'immagine. Gli istogrammi sono visualizzati graficamente come semplici grafici dove l'asse orizzontale corrisponde ai valori r_k, mentre l'asse verticale ai valori di intensiá $h(r_k)$. La sogliatura dell'immagine in Figura 1.5 si ottiene dall'osservazione dell'istogramma, dove la distribuzione di intensitá dei pixel dell'oggetto e dello sfondo sono distinte. Generalmente la binarizzazione tramite soglia globale si esegue quando é presente un unico oggetto di interesse caratterizzato da uno sfondo uniforme. Generalmente le tecniche di soglia adattiva sono efficaci nel trattamento di immagini che presentano una elevata variazione di intensitá luminosa.

1.5.2 Trasformazioni Geometriche

Le trasformazioni geometriche elementari eseguibili su un'immagine sono le traslazioni, le rotazioni, i cambiamenti di scala, e le distorsioni (orizzontale e verticale). Queste vengono utilizzate prevalentemente in fase di preprocessing e nelle operazioni di computer graphics. Le trasformazioni, pur modificando le relazioni spaziali tra i pixel, sono vincolate in modo da preservare la continuitá delle linee e i rapporti reciproci di posizione e proporzione degli oggetti di scena. Consistono in particolare di due operazioni: la trasformazione spaziale delle coordinate, e l'interpolazione che assegna i valori di intensitá alle nuove posizioni ottenute dalla trasformazione:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & tx \\ a_{21} & a_{22} & ty \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v \\ w \\ 1 \end{bmatrix}$$

La trasfomazione affine appena riportata permette di deformare i rettangoli in parallelogrammi in base ai valori degli elementi della matrice. L'introduzione delle coordinate omogenee permettte anche di concatenare una sequenza di trasformazioni geometriche elementari in un'unica matrice, la quale é ottenuta dal prodotto (nell'ordine opportuno) delle singole matrici di trasformazione. Le trasformazioni possono essere di tipo:

$$(x, y) = T(v, w)$$

dove per ogni posizione (v,w) dei pixel dell'immagine sorgente viene calcolata la corrispondente posizione spaziale (x, y) dei pixel dell'immagine trasfomata. In questo caso alcune coordinate potrebbero non essere coperte da nessun

pixel di input, oppure potrebbero essere destinazione di piú pixel dell'immagine sorgente. Con una trasformazione inversa si evita il problema della copertura non uniforme. Tutte le posizioni spaziali dell'immagine di destinazione vengono visitate, e per ciascuna di esse si determinano le coordinate corrispondenti nell'immagine sorgente

$$(v, w) = T^{-1}(x, y)$$

1.5.3 Traslazione

La traslazione a corpo rigido di un'immagine consiste nello spostamento di ogni suo punto di una certa distanza lungo una direzione prestabilita. Considerando $f(v, w)$ come l'immagine sorgente e $g(x, y)$ come l'immagine traslata, si puó scrivere

$$g(x, y) = f(v + tx, w + ty)$$

dove tx e ty sono rispettivamente lo spostamento lungo l'asse orizzontale e verticale. Per eseguire una trasformazione geometrica si utilizza una funzione generica caratterizzata da una matrice di trasformazione di dimensione 2x3. Nel caso particolare di una traslazione, la matrice deve avere la seguente forma:

$$T = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \end{bmatrix}$$

1.5.4 Trasformazioni di scala

Si possono modificare le dimensioni di un'immagine moltiplicando ciascuna coordinata per un fattore di scala. Le equazioni delle coordinate sono generalmente

$$x = av$$

$$y = bw$$

con a e b reali e positivi. Per uno stesso fattore (a = b) maggiore di uno si ha un ingradimento dell'immagine (zooming), mentre per uno stesso fattore minore di uno si ha un rimpicciolimento (shrinking). Il ridimensionamento ($a \neq b$) si puó eseguire utilizzando una matrice di trasformazione 2x3 del tipo:

$$T = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \end{bmatrix}$$

1.5.5 Rotazioni

La rotazione di un'immagine si esprime, nell'equazione delle coordinate, come

$$(x, y) = (v \cos\theta - w \sin\theta, v \sin\theta + w \cos\theta)$$

dove θ é l'angolo di rotazione rispetto all'asse orizzontale. Si puó ruotare e ridimensionare l'immagine utilizzando la seguente funzione che calcola l'apposita matrice di trasformazione.

Capitolo 2

ANALISI DI IMMAGINI

Il secondo capitolo di tesi propone un approfondimento sulle immagini binarie, le piú importanti e utilizzate nella Computer Vision, in particolare sullo studio di queste e le relative proprietá geometriche che si possono estrarre.

2.1 Immagini Binarie

Le immagini binarie sono immagini che vengono descritte su due valori, indicati in genere con 0 e 1, che rappresentano rispettivamente il nero e il bianco.
Le immagini binarie, come giá visto, sono utilizzati in molte applicazioni poiché sono piú semplici da elaborare anche se sono una rappresentazione impoverita delle informazioni dell'immagine originale. Tuttavia, queste risultano utili dove tutte le informazioni necessarie possono essere fornite dalla sagoma dell'oggetto, e quando é possibile ottenere la sagoma di tale oggetto facilmente.

Alcuni esempi di applicazione di queste comprendono:

- Identificazione degli oggetti su un nastro trasportatore (per esempio, lo smistamento cioccolatini)
- Individuare orientazione di oggetti
- Interpretazione del testo.

Spesso il risultato di tecniche di elaborazione delle immagini é rappresentato sotto forma di un'immagine binaria, per esempio, il rilevamento dei bordi puó essere un'immagine binaria (punti di bordo e non punti di bordo). Le tecniche di elaborazione binaria delle immagini possono essere utili per la

successiva elaborazione di queste immagini appena costruite.

Le immagini binarie sono in genere ottenute tramite sogliatura di un'immagine a livelli di grigio. I pixel con un livello di grigio al di sopra della soglia vengono impostati a 1 (equivalente 255), mentre gli altri vengono impostati a 0. Questo produce un oggetto bianco su fondo nero (o viceversa, a seconda dei valori relativi di grigio dell'oggetto e dello sfondo). Naturalmente, il negativo di un'immagine binaria é anch'essa un'immagine binaria ottenuta semplicemente con i valori dei pixel invertiti.

La scelta di una soglia puó risultare difficile, e molti approcci si basano sull'uso dell'istogramma di livello di grigio dell'immagine originaria. In caso di istogramma bimodale la scelta di soglia manuale risulterá facile con la possibilitá di poter costruire anche una procedura automatica per la sua determinazione. Idealmente, in presenza di oggetto nero su fondo bianco l'istogramma ottenuto sará simile a quello di Figura 2.1.

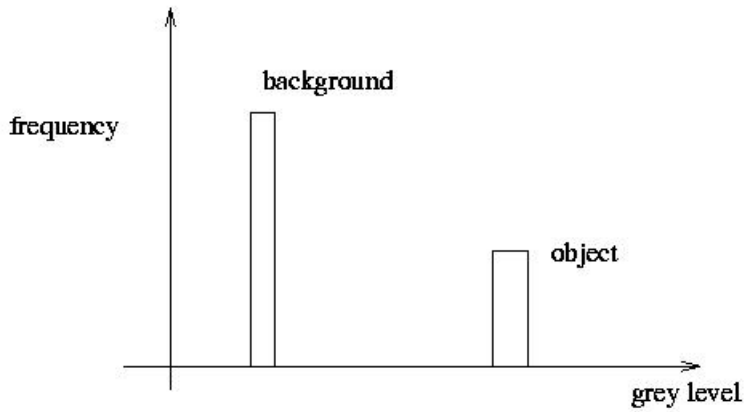


Figura 2.1: Istogramma ideale di oggetto luminoso su sfondo scuro

In presenza di rumore l'istogramma che si ottiene avrá figura del tipo:

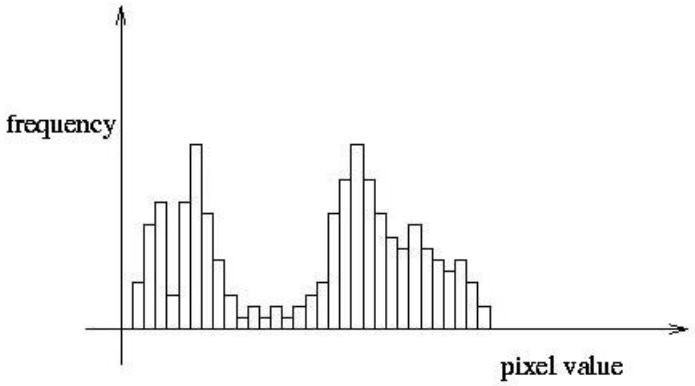


Figura 2.2: Frequenza di ciascun valore di pixel relativi alla scala di grigio

Se i livelli di grigio di oggetto e lo sfondo sono abbastanza vicini, l'influenza del rumore puó provocare un oggetto visualizzato molto sporco e spesso anche compromesso. Il risultato dell'istogramma sará dovuto alle componenti di rumore sommate a quelle dell'immagine. In questo caso risulterá molto piú difficile determinare un valore di soglia per la digitalizzazione dell'immagine stessa. In questo caso l'istogramma non sará piú bimodale e non ci sará modo chiaro di scegliere la soglia. Con osservazioni ripetute e con calcolo della media si puó ottenere qualcosa, ma la diffusione del istogramma sará dovuta anche al cambiamento di luminositá o alle variazioni di colore sullo sfondo e l'oggetto.

2.2 Analisi su Immagini Binarie

2.2.1 Analisi tramite Momenti

Una prima analisi delle immagini binarie consiste nel trovare i vari attributi degli oggetti e della scena con l'obiettivo di utilizzarli per identificare gli oggetti presenti e per determinarne la loro posizione e/o orientamento.
Considerando un immagine binaria di tipo continuo (cioé con una risoluzione infinita) vengono ora definiti e studiati i momenti di un immagine.
Nell'elaborazione delle immagini e nella visione artificiale il momento di un'immagine, in analogia con il concetto generale di momento, viene definito come una particolare media dell'intensitá dei pixel che compongono l'immagine. In senso piú generale, sono detti momenti anche le funzioni di tali medie, che godono di particolari proprietá o caratteristiche.

Il momento di un immagine (o momento bidimensionale geometrico) di ordine $(p+q)$ di una funzione $f(x,y)$ viene definito come:

$$M_{pq} = \int_{a_1}^{a_2} \int_{b_1}^{b_2} x^p y^q f(x,y) dxdy$$

dove p,q=0,1,2...
Da notare che il prodotto $x^p y^q$ risulta la funzione basilare per il calcolo del momento. Un gruppo di momenti di dimensione n viene rappresentato come M_{pq} dove p + q $\leq$ n e contiene $1/2(n+1)(n+2)$ elementi.

Per il teorema di unicitá se $f(x,y)$ é continua a tratti ed ha valori non nulli solo in una porzione finita del piano xy, allora esistono i momenti di ogni ordine e la sequenza dei momenti (M_{pq}) é unicamente determinata da f(x,y). Rispettivamente (M_{pq}) determina unicamente f(x,y). In pratica, la funzione puó essere descritta in funzione dei suoi momenti di ordine piú basso. Adattando questa definizione ad un'immagine digitale i cui pixel sono caratterizzati da intensitá I(x,y), il momento semplice M_{ij} viene dato da:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y)$$

Vengono ora definiti i vari momenti principali dell'analisi delle immagini binarie al fine di dare una minima base teorica su questi e sui loro possibili utilizzi.
Il momento base o M_{00} di una funzione $f(x,y)$

$$M_{00} = \int_{a_1}^{a_2} \int_{b_1}^{b_2} f(x,y) dxdy$$

rappresenta la massa totale della funzione considerata.
Nel caso della computer vision e dell'analisi delle immagini binarie rappresenta l'area totale dell'immagine.
I successivi momenti di ordine minore rappresentano le proprietá geometriche fondamentali alla base dell'analisi delle immagini binarie. I momenti M_{10} e M_{01} definiti come segue

$$M_{10} = \int_{a_1}^{a_2} \int_{b_1}^{b_2} x \cdot f(x,y) dxdy$$

$$M_{01} = \int_{a_1}^{a_2} \int_{b_1}^{b_2} y \cdot f(x,y) dxdy$$

e rappresentano rispettivamente il centro di massa della funzione in coordinate x e y. Il centro di massa viene definito come il punto dove tutta la massa dell'immagine dovrebbe essere concentrata senza cambiare il primo momento dell'immagine rispetto ad ogni asse.
Nel caso delle immagini binarie la posizione del punto si ottiene con

$$\overline{x} = \frac{M_{10}}{M_{00}}$$

$$\overline{y} = \frac{M_{01}}{M_{00}}$$

Le coordinate appena definite possono essere utilizzate come definizione della posizione dell'immagine.
Il momento centrale di un immagine viene definito come:

$$\mu_{pq} = \int_{a_1}^{a_2} \int_{b_1}^{b_2} (x - \overline{x})^p \cdot (y - \overline{y})^q f(x, y) dx dy$$

in riferimento ai valori di centro di massa definiti appena sopra.
I momenti centrali μ_{pq} definiti nell'equazione sopra sono invarianti alla traslazione delle coordinate

$$x' = x + \alpha$$

$$y' = y + \beta$$

dove α e β sono delle costanti.
I momenti del secondo ordine, conosciuti anche come momenti d'inerzia posso essere usati per determinare importanti caratteristiche dell'immagine quali l'orientazione. In generale l'orientazione di un immagine rappresenta un valore di angolo rispetto alle direzioni degli assi principali. Vengono ora elencati i momenti centrali fino al terzo ordine in particolare facendo riferimento al calcolo dell'orientazione, cosa di particolare importanza nell'ambito dell'analisi delle immagini e nelle possibili applicazioni di queste.

$$\mu_{00} = M_{00}$$

$$\mu_{01} = 0$$

$$\mu_{10} = 0$$

$$\mu_{11} = M_{11} - \overline{x} \cdot M_{01} = M_{11} - \overline{y} \cdot M_{10}$$

$$\mu_{20} = M_{20} - \overline{x} \cdot M_{10}$$

$$\mu_{02} = M_{02} - \overline{y} \cdot M_{01}$$
$$\mu_{21} = M_{21} - 2\overline{x} \cdot M_{11} - \overline{y} \cdot M_{20} + 2\overline{x}^2 \cdot M_{01}$$
$$\mu_{12} = M_{12} - 2\overline{y} \cdot M_{11} - \overline{x} \cdot M_{02} + 2\overline{y}^2 \cdot M_{10}$$
$$\mu_{30} = M_{30} - 3\overline{x} \cdot M_{20} + 2\overline{x}^2 \cdot M_{10}$$
$$\mu_{03} = M_{03} - 3\overline{y} \cdot M_{02} + 2\overline{y}^2 \cdot M_{01}$$

L'orientazione dell'immagine si puó facilmente ottenere da:

$$\theta = \frac{1}{2} \tan^{-1} \frac{2\mu_{11}}{\mu_{20} - \mu_{02}}$$

In riferimento all'orientazione dell'immagine, tale proprietá sará ripresa nei successivi paragrafi e capitoli, in riferimento al suo utilizzo nella libreria cvBlobs.

2.3 Filtri su Immagini Binarie

Nell'utilizzo di software di elaborazione delle immagini l'implementazione di operazioni quali miglioramento, sfocatura, rilevamento contorni sono le attivitá principali che riguardano la computer vision. Queste sono solo alcune delle numerose operazioni realizzabili con dei filtri. La progettazione e l'utilizzo dei filtri é strettamente connessa allo studio di segnali e sistemi.
In questo caso la grandezza da studiare é la quantitá di colore di ogni pixel. E' facile intuire che si tratta di un segnale bidimensionale discreto e soprattutto che la variabile di riferimento non é il tempo, ma lo spazio: l'interesse riguarda infatti le variazioni spaziali del colore. Un sistema é un dispositivo che, dato un segnale in ingresso, lo elabora e ne fornisce il risultato in uscita. Un sistema é in genere una modellazione matematica di un fenomeno fisico. Esistono numerose classificazioni dei sistemi, ma non si entrerá nel merito. Ci si accontenta di sapere che anche i sistemi possono essere continui o discreti (o talvolta ibridi).
Sapendo la risposta impulsiva di un sistema e conoscendo l'espressione del segnale in ingresso, l'uscita del sistema puó esser calcolata tramite l'operazione di convoluzione definita nel continuo da:

$$v(t) = u(t) * h(t) = \int_{-\infty}^{+\infty} u(t)h(t-\tau)dt = \int_{-\infty}^{+\infty} u(t-\tau)h(t)dt$$

e nel discreto da:

$$v(t) = u(t) * h(t) = \sum_{-\infty}^{+\infty} u(t)h(t-\tau) = \sum_{-\infty}^{+\infty} u(t-\tau)h(t)$$

Operando con un segnale discreto e bidimensionale, l'immagine in uscita viene calcolata con la seguente operazione:

$$v(x,y) = u(x,y) * h(x,y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} u(m,n)h(x-m, y-n)$$

dove M e N sono larghezza e altezza dell'immagine mentre il sistema é il filtro che si vuole applicare.
Il filtro é un sistema che viene utilizzato solitamente per selezionare le frequenze di interesse ovvero il numero di variazioni di colore per unitá spaziale. Come si puó immaginare, sia il segnale che il sistema sono in realtá delle matrici bidimensionali, quindi l'operazione che stiamo per eseguire é una convoluzione fra matrici. La matrice che identifica il sistema é detta matrice o maschera di convoluzione o kernel ed é in genere una matrice quadrata di ordine dispari. La convoluzione fra matrici mira a calcolare il nuovo valore di ogni pixel sovrapponendo ad ognuno di essi la matrice kernel (con il centro sul punto in questione) ed eseguendo la sommatoria dei prodotti fra ogni pixel ed il corrispettivo elemento nella matrice di convoluzione.

Esempio. Si suppone di avere una matrice di convoluzione come la seguente:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -5 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

e si considera ora un pezzo di una matrice di un immagine a caso:

$$\begin{bmatrix} .. & .. & .. & .. & .. \\ .. & 124 & 112 & 98 & .. \\ .. & 124 & 45 & 29 & .. \\ .. & 110 & 102 & 110 & .. \\ .. & .. & .. & .. & .. \end{bmatrix}$$

Il nuovo valore del pixel centrale (di valore 45) sará:

$$0 \cdot 124 + 1 \cdot 112 + 0 \cdot 98 + 1 \cdot 124 + 45 \cdot (-5) + 1 \cdot 29 + 0 \cdot 110 + 1 \cdot 102 + 0 \cdot 110 = 142$$

Lo stesso procedimento sará applicato ad ogni pixel.
Per tutti i pixel sul confine (e nel caso di matrici di ordine superiore al terzo

anche per elementi piú interni) alcuni elementi della matrice di convoluzione si trovano nell'area al di fuori dell'immagine.
In questi casi si puó agire su 3 vie:

- si ignora la parte esterna al kernel;
- la parte del kernel esterna all'immagine verrá applicata ai pixel del bordo opposto cosicché i pixel che scompaiono da un lato ricompaiono dall'altro;
- si sceglie di non applicare la matrice ai bordi.

La successiva parte del capitolo introduce i principali filtri applicabili su immagini binarie e i metodi per l'estrazione delle informazioni da queste.

2.3.1 Estrazione di Feature da Immagini Binarie

Le immagini digitali acquisite dai sistemi di visione dipendono dalla prospettiva e dalla illuminazione perció le cause comuni per la discontinuitá in una immagine sono dovute ai contorni delle superfici, alle ombre e alle occlusioni. Possiamo definire formalmente una feature come qualsiasi discontinuitá o uniformitá che sia significativa e utile. Uno dei piú grandi vantaggi dell'estrazione di feature sta nella significativa riduzione dell'informazione (comparata all'immagine originale) nella rappresentazione di una immagine allo scopo di capirne il contenuto. La molteplicitá di feature contenute in una immagine, sommate alla presenza di rumore, rendono arduo il compito di progettare un filtro che sia in grado di isolare una specifica caratteristica dalla scena. In genere se la qualitá dell'immagine non é buona é necessario ridurre il rumore e successivamente esaltare la caratteristica di interesse attraverso un filtro specifico chiamato feature (o keypoint) detector che risponde in modo diverso a caratteristiche diverse. A questo punto é possibile avviare un processo decisionale per la selezione delle caratteristiche desiderate.
Una possibile distinzione tra tipi di features estratte é quella tra General Features e Domain Features. Le prime sono dette descrittori di basso livello e sono indipendenti dal contesto dell'immagine. Le Domain Features invece dipendono dal contesto dell'immagine e sono utilizzate prevalentemente per il riconoscimento di volti (face recognition) e per l'analisi delle impronte digitali (fingerprint identification). Qualora si voglia identificare questo tipo di descrittori é necessaria l'analisi delle features di basso livello, per tale motivo le Domain Features sono dette di alto livello.

I descrittori di basso livello possono essere in generale classificati in 3 tipi:

- Blob. Descrittori di livello zero, costituito da regioni uniformi, caratterizzate da ridotte variazioni dell'intensitá.
- Edge (spigolo). Descrittore di primo livello che si ha in corrispondenza di discontinuitá del gradiente monodirezionali.
- Point features, o corner, sono punti distinti dell'immagine localizzati in corrispondenza di discontinuitá bidirezionali del gradiente dell'intensitá luminosa. A seconda della forma assunta dal corner si distinguono giunzioni a T, Y, X o L.

2.3.2 Edge Detection

Per la rilevazione di feature di alto livello é fondamentale la segmentazione dell'immagine per la quale l'edge detection gioca un ruolo molto importante. Un edge puó essere considerato come un confine tra due regioni dissimili ed essenzialmente il contorno di un oggetto corrisponde ad un brusco cambiamento nei livelli di intensitá. L'output dell'edge detection dovrebbe essere una mappa dei bordi nella quale il valore di ogni pixel riflette quanto sono verificati i requisiti per essere parte di un bordo da parte del pixel corrispondente nell'immagine originale.

Rilevazione dei bordi mediante derivata prima
Per rilevare la posizione di un bordo spesso viene utilizzata la derivata del primo ordine, che é nulla in presenza di un segnale costante, mentre dovrebbe presentare il massimo in presenza di un edge. La funzione di magnitudine del gradiente in un'immagine digitalizzata puó essere cosi formulata:

$$d(x, y) = \sqrt{\Delta x^2 + \Delta y^2}$$

con

$$\Delta x = I(x + 1, y) - I(x, y)$$
$$\Delta y = I(x, y + 1) - I(x, y)$$

Spesso nell'image processing la derivata prima di un'immagine digitale viene effettuata attraverso la convoluzione con una maschera chiamata edge operator.
Per rilevare un edge in una immagine reale é opportuno effettuare una preliminare riduzione del rumore, dopodiché si effettua la convoluzione con l'edge

operator e infine dall'output si eliminano i valori non corrispondenti ai massimi.
Le maschere piú semplici per effettuare una derivata del primo ordine sono costituite da un vettore [1,0,-1] per rilevare bordi orizzontali e dal suo trasposto per bordi verticali.

$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Uno dei primi operatori utilizzati fu il Robert's Cross Operator che utilizza una maschera 2x2 , ma poiché la derivata prima esalta il rumore, analogamente ad un filtro passa-alto, i successivi edge operator sono costituiti da matrici 3x3 in modo da includere all'interno anche il neighborhood averaging.
Seguendo tale principio é stato ideato il filtro di Prewitt e il filtro di Sobel dove quest'ultimo peró pesa in modo differente i pixel in base alla distanza dal pixel centrale.

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Le matrici sopra rappresentano il filtro di Prewitt, mentre quelle seguenti quello di Sobel, entrambi i filtri mostrano degli output frammentati seppur ben visibili. I due filtri appena citati saranno ripresi piú nel dettaglio nei successivi paragrafi.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Rilevazione dei bordi mediante derivata seconda
Un'alternativa alla ricerca dei massimi della derivata del primo ordine é quella di trovare l'attraversamento dello zero nella derivata del second'ordine. La

derivata seconda puó essere approssimata con la differenza tra le derivate del primo ordine di due pixel adiacenti:

$$f''(x) = \Delta_x - \Delta_{x+1}$$

dove considerando il valore di Δ_x prima definito:

$$f''(x) = -\Delta_x + 2\Delta_{x+1} - \Delta_{x+2}$$

Un operatore che utilizza la derivata del secondo ordine é il Laplaciano che sfrutta le maschere, rappresentate nell'illustrazione seguente, ottenute dalla combinazione del vettore [-1, 2, -1] in orizzontale, verticale e diagonale.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Il laplaciano, non essendo utile per stabilire la direzione di un edge, viene utilizzato in genere per stabilire la posizione di uno spigolo che si fa coincidere con lo zero crossing. Inoltre questo filtro non viene utilizzato per il rilevamento dei bordi nella sua forma originale, sia perché eccessivamente sensibile al rumore, sia per il fatto che produce dei doppi bordi che complicano la segmentazione.
In genere si effettua il Laplaciano della Gaussiana (Figura 2.3) ottenuto facendo, preliminarmente al Laplaciano, uno smoothing Gaussiano per ridurre il rumore e neutralizzare l'effetto di amplificazione del rumore causato dalla derivata seconda.

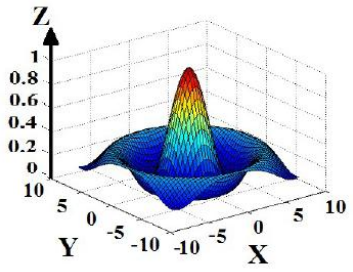


Figura 2.3: Gaussiana

La maschera (matrice) utilizzata che approssima la forma della gaussiana é:

$$\begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

2.3.3 Filtro di Sobel

L'algoritmo di Sobel ricerca punti validi di contorno nei massimi locali della derivata prima della funzione da analizzare. Per assorbire le alterazioni della luminositá introdotte da rumore e distorsioni del sistema di acquisizione delle immagini, si stabilisce una soglia (cutoff o threshold) che puó essere calcolata in modo automatico o impostata direttamente dall'operatore. Un metodo molto semplice per stabilire il valore della soglia puó essere quello di valutare il rapporto tra la somma delle intensitá di ogni pixel ed il numero dei pixels costituenti la matrice immagine. Un'altra peculiaritá del valore di soglia é che esso puó assumere un valore unico per l'intera immagine oppure adattarsi localmente.

I filtri utilizzati dall'operatore di Sobel per estrarre il gradiente sono i seguenti:

$$F_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$F_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

dove F_x estrae la componente orizzontale del gradiente, F_y quella verticale.
Come si puó facilmente notare viene dato un peso maggiore agli elementi che stanno sopra ed accanto all'elemento da analizzare. Per capire il funzionamento del filtro prendiamo come esempio una matrice quadrata delle stesse dimensioni dei filtri (l'elemento x é quello da analizzare).

$$M = \begin{bmatrix} a & b & c \\ d & x & e \\ f & g & h \end{bmatrix}$$

L'azione di ogni filtro é quella di mascherare ogni elemento della matrice da analizzare con il peso relativo. Immediatamente si puó dedurre che i contorni non possono essere determinati in nessuno dei punti che appartengono alla regione piú esterna dell'immagine (in pratica la prima e l'ultima riga, la prima e l'ultima colonna). Il gradiente nella direzione desiderata si ottiene semplicemente sommando gli elementi della matrice filtrata.
In particolare le matrici filtrate da F_x e F_y risultano:

$$M_x = \begin{bmatrix} -1 \cdot a & 0 \cdot b & 1 \cdot c \\ -2 \cdot d & 0 \cdot x & 2 \cdot e \\ -1 \cdot f & 0 \cdot g & 1 \cdot h \end{bmatrix}$$

$$M_y = \begin{bmatrix} -1 \cdot a & -2 \cdot b & -1 \cdot c \\ 0 \cdot d & 0 \cdot x & 0 \cdot e \\ 1 \cdot f & 2 \cdot g & 1 \cdot h \end{bmatrix}$$

mentre i rispettivi gradienti:

$$Gx = -1 \cdot a - 2 \cdot d - 1 \cdot f + 1 \cdot c + 2 \cdot e + 1 \cdot h$$

$$Gy = -1 \cdot a - 2 \cdot b - 1 \cdot c + 1 \cdot f + 2 \cdot g + 1 \cdot h$$

Ora basta calcolare il gradiente come:

$$G = |G_x| + |G_y|$$

e confrontare il valore ottenuto con la soglia. Se il gradiente risulterá essere maggiore della soglia allora il punto rispetto al quale é stato calcolato il gradiente é un punto di contorno, altrimenti no.

2.3.4 Filtro di Prewitt

L'algoritmo di Prewitt si basa sulle stesse considerazioni dell'algoritmo di Sobel, anzi ne é stato per certi versi un precursore. La differenza fondamentale che c'é fra i due filtri é che mentre i kernel di Sobel sono pesati sia in orizzontale che in verticale, quelli di Prewitt non fanno alcuna differenza tra gli elementi adiacenti al pixel che viene analizzato.
I filtri utilizzati dall'operatore di Prewitt per estrarre il gradiente sono i seguenti:

$$F_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

$$F_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Come nel filtro precedente F_x estrae la componente orizzontale del gradiente, F_y quella verticale.
Come si puó facilmente notare non c'é nessuna differenza di peso tra gli elementi che compongono il filtro. Per capire il funzionamento del filtro prendiamo come esempio una matrice quadrata delle stesse dimensioni dei filtri (l'elemento x é sempre quello da analizzare).

$$M = \begin{bmatrix} a & b & c \\ d & x & e \\ f & g & h \end{bmatrix}$$

L'azione di ogni filtro é quella di mascherare ogni elemento della matrice da analizzare con il peso relativo. Il gradiente nella direzione desiderata si ottiene semplicemente sommando gli elementi della matrice filtrata. In particolare le matrici filtrate da F_x e F_y risultano:

$$M_x = \begin{bmatrix} -1 \cdot a & 0 \cdot b & 1 \cdot c \\ -2 \cdot d & 0 \cdot x & 2 \cdot e \\ -1 \cdot f & 0 \cdot g & 1 \cdot h \end{bmatrix}$$

$$M_y = \begin{bmatrix} -1 \cdot a & -2 \cdot b & -1 \cdot c \\ 0 \cdot d & 0 \cdot x & 0 \cdot e \\ 1 \cdot f & 2 \cdot g & 1 \cdot h \end{bmatrix}$$

mentre i rispettivi gradienti sono:

$$Gx = -1 \cdot a - 1 \cdot d - 1 \cdot f + 1 \cdot c + 1 \cdot e + 1 \cdot h$$

$$Gy = -1 \cdot a - 1 \cdot b - 1 \cdot c + 1 \cdot f + 1 \cdot g + 1 \cdot h$$

Ora basta calcolare il gradiente come:

$$G = \sqrt{G_x^2 + G_y^2}$$

e confrontare il valore ottenuto con la soglia. Se il gradiente risulterá essere maggiore della soglia allora il punto rispetto al quale é stato calcolato il gradiente é un punto di contorno, altrimenti no.

2.3.5 Filtro di Canny

Nel 1986 lo statunitense John F. Canny progettó un algoritmo per il riconoscimento dei contorni che viene ormai definito come lo standard in questo campo, per questo motivo ne viene offerta un'analisi dettagliata preferendolo ad altri algoritmi che nella maggioranza dei casi non offrono risultati migliori. L'algoritmo si divide in quattro fasi:

1. *Riduzione del rumore*
2. *Ricerca del gradiente di luminositá dell'immagine*
3. *Soppressione dei non-massimi*
4. *Individuazione dei contorni mediante sogliatura con isteresi*

1) Riduzione del rumore
Un problema primario negli algoritmi di riconoscimento dei contorni é dato dalla presenza di rumore nelle immagini non processate, per cui come primo passo si applica all'immagine un filtro spaziale, tramite il processo di convoluzione giá visto, il cui scopo é rimuovere le alte frequenze su cui il rumore interferisce in maniera piú problematica. Per far ció viene scelto un filtro gaussiano il cui valore di deviazione standard puó essere modificato in modo da prediligere il riconoscimento di bordi piccoli e netti oppure piú grandi e graduali (nell'implementazione che ne fornisce Matlab ha un valore di deviazione standard predefinito $\sigma = 1$, mentre la dimensione del filtro viene scelta automaticamente a seconda del valore di σ, dato che quest'ultimo ne definisce la velocitá con cui i valori tendono a 0) ed il risultato di questa prima fase é un'immagine leggermente sfuocata, in cui nessun pixel é affetto dal rumore in maniera significativa.

2) Ricerca del gradiente della luminositá dell'immagine
La definizione di gradiente, in particolare i gradienti calcolati sull'asse x ed y, (Gx,Gy) si riportano alle definizioni:

$$G_x(x,y) = \frac{dF(x,y)}{dx}$$

$$G_y(x,y) = \frac{dF(x,y)}{dy}$$

$$G = \sqrt{G_x^2 + G_y^2}$$

Nella maggior parte delle implementazioni i valori dei gradienti nelle due dimensioni vengono calcolati utilizzando l'operatore di Sobel visto in precedenza:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Questo operatore garantisce una discreta approssimazione pur mantenendo bassa la richiesta di calcolo. Per migliori approssimazioni sarebbe opportuno utilizzare differenti tecniche che peró aumentano enormemente la quantitá di tempo necessario per processare un'immagine e necessitano del calcolo di numeri irrazionali.
Si definisce θ l'angolo di direzione del contorno:

$$\theta = arctan\frac{G_y}{G_x}$$

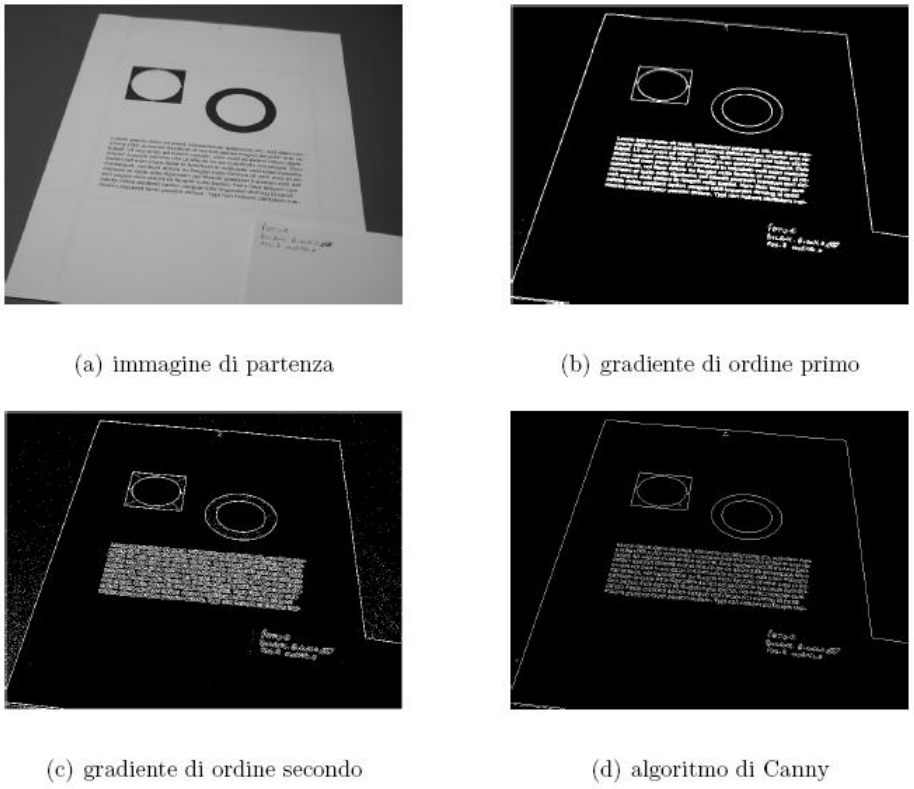
Il valore di θ viene arrotondato ai valori di 0, 45, 90 e 135 rappresentanti rispettivamente i bordi verticali, orizzontali e lungo le due diagonali.
Questa fase restituisce per ogni pixel la direzione, arrotondata, di massimo gradiente con il valore di quest'ultimo.

3) Soppressione dei non-massimi
In questa fase si azzerano i valori dei pixel non considerati parte del contorno, cioé i pixel il cui valore di intensitá non é maggiore di quello dei pixel adiacenti situati lungo la direzione data dal valore θ in quel punto. Il risultato é un'immagine binaria con una linea sottile in corrispondenza dei bordi degli oggetti nell'immagine.

4) Individuazione dei contorni mediante sogliatura con isteresi
Una debolezza riguardante gli algoritmi di riconoscimento di contorni é data dal valore di sogliatura da assegnare per ottenere un buon risultato. Questo valore, viene spesso determinato empiricamente, cercando di ottenere il miglior risultato possibile, ma in generale non si puó prevedere quale sia il valore minimo che debba avere il gradiente affinché il pixel considerato sia parte del contorno. Per attenuare questa debolezza si utilizza un metodo di sogliatura con isteresi, che prevede due valori: il valore di soglia alta e quello di soglia bassa. Ogni punto il cui gradiente sia superiore al valore di soglia alta é automaticamente definito parte del contorno (ne viene assegnato il

valore 1), inoltre ogni punto contiguo ad un punto del contorno che abbia un valore del gradiente superiore al valore di soglia bassa entra a far parte del contorno, assegnando cosi valore 0 a tutti i punti rimanenti e quelli al di sotto della soglia bassa. Il risultato é l'immagine binaria dei contorni.

Nella figura seguente viene mostrato il risultato dell'applicazione di differenti algoritmi di riconoscimento dei contorni. La (a) mostra l'immagine originale, mentre le (b) e (c) sono ottenute mediante gradiente di, rispettivamente, primo e secondo ordine. Infine nella figura (d) viene mostrato il risultato dell'applicazione dell'algoritmo di Canny che da un'analisi qualitativa risulta decisamente migliore rispetto ai precedenti.



(a) immagine di partenza

(b) gradiente di ordine primo

(c) gradiente di ordine secondo

(d) algoritmo di Canny

Figura 2.4: Filtri a confronto

2.3.6 Filtro di Gabor

Un filtro di Gabor é un filtro lineare ottenuto mediante la modulazione di una funzione sinusoidale con una funzione Gaussiana di deviazione standard σ. L'applicazione di questo filtro alle immagini, viste come un segnale bidimensionale, viene cosi composto: una funzione sinusoidale con orientazione a 30 gradi rispetto all'asse delle x moltiplicato ad un kernel gaussiano. La Figura 2.5 mostra in a) la funzione sinusoidale, in b) il kernel Gaussiano e in c) il risultato:

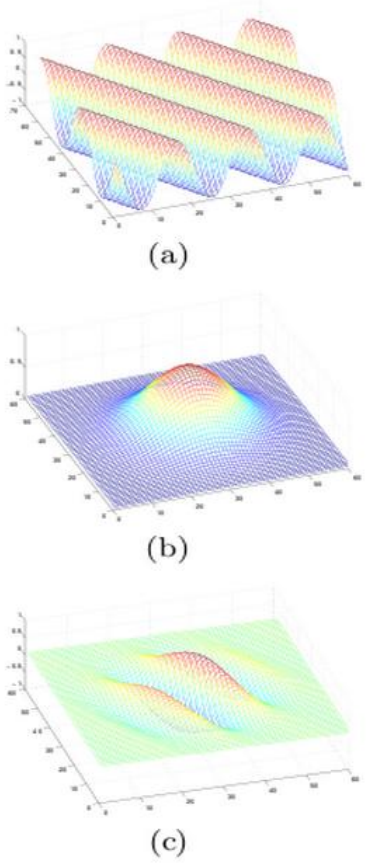


Figura 2.5: Realizzazione filtro di Gabor

Dato il Filtro di Gabor $g(x, y, \theta, \phi)$ centrato nell'origine; x e y rappresentano le coordinate spaziali dell'immagine ed i parametri θ e ϕ rispettivamente la scala e l'orientazione. Di seguito si presenta la formula del filtro:

$$g(x, y, \theta, \phi) = exp(-\frac{x^2 + y^2}{\sigma^2})exp(2\pi\theta i(x\cos\phi + y\sin\phi))$$

Piú specificatamente il parametro θ é la frequenza spaziale della funzione sinusoidale. Il suo valore é specificato in pixel. Valori validi sono numeri maggiori uguali di 2. Per prevenire a inconvenienti indesiderati che si potrebbero presentare ai bordi dell'immagine, la scala dovrebbe essere piú piccola di un quinto della grandezza dell'immagine di input.
Il valore di ϕ specifica l'orientazione della funzione sinusoidale; il suo valore é espresso in gradi. I valori validi sono compresi tra 0 e 360 gradi.
La relazione tra σ e θ é $\sigma = 0.5 \cdot 1/\theta$ dove $1/\theta$ rappresenta la lunghezza d'onda del fattore coseno ed é inversamente proporzionale alla frequenza.
Fissate le dimensioni n x n, viene calcolato un filtro di Gabor con l'equazione riportata sopra, il risultato é visualizzato come due immagini rappresentanti la parte reale (a sinistra) e la parte immaginaria del filtro (a destra). Il filtro viene creato con l'origine al centro dell'immagine.

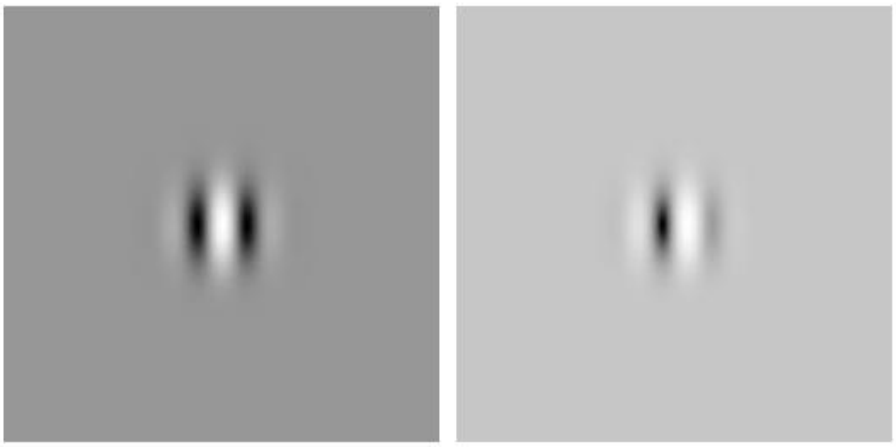


Figura 2.6: Filtro di Gabor reale e immaginario

Per il task di image retrieval é necessario calcolare le feature estratte dall'immagine filtrata. E' inusuale realizzare un unico filtro di Gabor con una sola orientazione e una sola scala per la creazione delle feature. Risulta utile realizzare un banco di filtri di Gabor, In base al numero di orientazioni e il numero di scale desiderati si realizzano N filtri differenti. Ad esempio se le orientazioni sono 6 e le scale sono 4, il risultato sará un banco caratterizzato da 6x4 filtri.
Sfruttando le proprietá del Teorema di convoluzione nel dominio di Fourier e le proprietá di traslazione, il filtro reale o immaginario di Gabor viene applicato all'immagine mediante prodotto punto punto tra la FFT dell'immagine originale e la FFT del filtro e seguito dall'antitrasformata di Fourier del ri-

sultato. Come si puó notare nelle immagini seguenti l'immagine al centro é l'applicazione del filtro a 90 gradi che enfatizza le linee verticali, mentre nell'immagine a destra vengono enfatizzate quelle orizzontali poiché il filtro ha come orientazione 180 gradi.

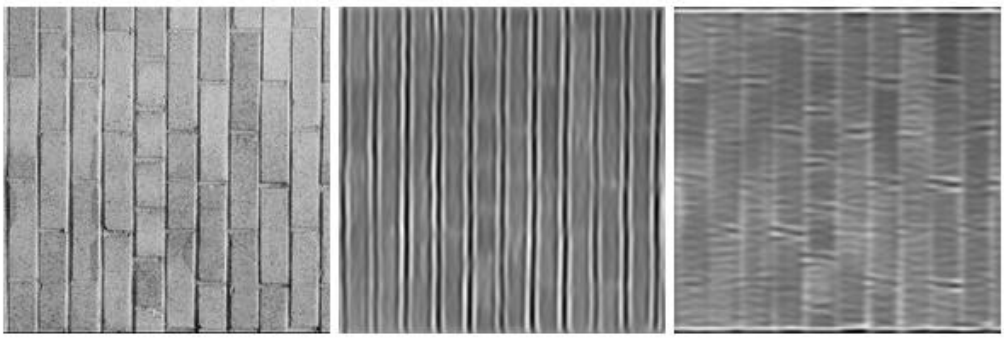


Figura 2.7: Esempio di Filtraggio

Capitolo 3

SVILUPPO SISTEMA DI VISIONE

Il capitolo seguente prevede una discussione sulla realizzazione del sistema di visione e una descrizione delle procedure di base che compongono lo stesso.

3.1 Obiettivi e sintesi di progetto

Il sistema di visione é stato interamente sviluppato con Microsoft Visual Studio 2008, sia nella sua parte di interfacciamento con utente che la parte di algoritmo per i vari calcoli. Dopo le varie introduzioni viste nei capitoli precedenti risulta ora opportuno andare a spiegare nel dettaglio in cosa consiste il lavoro e i vari obiettivi del sistema.
Il progetto prevede come argomento principale la visione computazionale applicata alla robotica. L'ambiente di lavoro, che sará descritto piú nel dettaglio nei successivi capitoli, prevede un robot Adept Four parallelo con applicato un supporto con profilo a C all'organo terminale sul quale sará fissato un recipiente contenente del liquido. Tramite delle telecamere (la configurazione principale ne prevede solamente una con visione sullo spostamento rispetto all'asse x del sistema di riferimento del robot) si é in grado di sondare l'ambiente e lo spazio di lavoro. La telecamera risulta comunque posizionata in modo da catturare solamente una piccola porzione dello spazio di lavoro, larga circa 50cm e al livello del recipiente contenente il liquido. Tutta questa configurazione viene realizzata per un controllo sulle oscillazioni del liquido stesso. Tramite telecamere infatti é possibile acquisire in tempo reale dei fotogrammi sui quali verrá poi applicato un algoritmo per l'estrazione dell'angolo di inclinazione del fluido. Il seguito a movimentazioni infatti il liquido (indipendentemente dal recipiente in cui é contenuto) comincerá ad

oscillare. Il programma di calcolo in tempo reale dovrá estrapolare, secondo metodi e criteri diversi, l'angolo di oscillazione del liquido rispetto alla sua posizione di quiete.

3.2 Struttura di Sistema

Dopo una breve introduzione al problema, che sará comunque rivisto piú nel dettaglio successivamente, viene ora proposta la configurazione dell'organizzazione del codice utilizzato, comprensivo di parte grafica e di calcolo.
Il codice sviluppato prevede una divisione delle due attivitá di lavoro (grafica e di calcolo) al fine di incrementarne le prestazioni e l'utilizzo. Una parte di codice prevede la realizzazione della parte di interfaccia con l'utente, la quale si baserá su una libreria .dll ottenuta tramite codice secondario e importato nel primo. La parte di grafica infatti sará sviluppata a parte e conterrá (oltre ai vari possibili eventi gestiti dall'utente) i collegamenti e l'importazione della libreria di calcolo. La parte di algoritmo per il calcolo di profilo del liquido e dell'angolo di oscillazione é scritta totalmente in codice C (con supporto delle librerie OpenCV per la trattazione delle immagini acquisite da telecamera) e prevede la creazione della libreria che dovrá poi essere importata e utilizzata in esecuzione dalla parte grafica del sistema. Viene ora riportato un grafico in cui é ben visibile la struttura del progetto e i relativi file con importazioni.

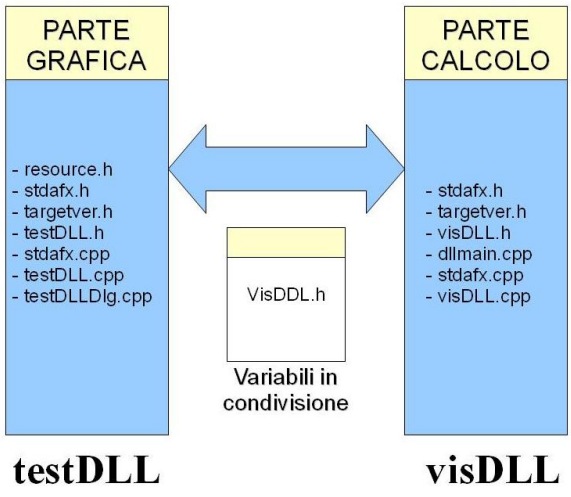


Figura 3.1: Struttura di Progetto

3.2.1 Struttura parte interfaccia

La parte di interfaccia é formata, come ogni altro codice in C,VisualC,C++ etcetc, dalla parte di intestazione, ovvero la parte di header, dalla parte di codice gestione eventi in cpp e la parte di grafica pura per la realizzazione dell'interfaccia stessa. La parte di header prevede un totale di 5 file in cui vengono definite le variabili globali (in comune con la parte esterna di calcolo) e i file necessari per l'implementazione del lavoro tramite thread. Il progetto di interfaccia prevede poi 3 file di gestione degli eventi, relativi ai pulsanti e check box per la selezione del tipo di algoritmo e di visualizzazione da usare durante l'esecuzione del programma finale. In appendice di tesi verrá riportato il codice principale per la gestione degli eventi e una breve spiegazione di questo.

Lista file di intestazione:

1. *resource.h*
2. *stdafx.h*
3. *targetver.h*
4. *testDLL.h*
5. *testDLLDlg.h*

Lista file di origine:

1. *stdafx.cpp*
2. *testDLL.cpp*
3. *testDLLDlg.cpp*

Lista file di risorse:

1. *testDLL.rc*

3.2.2 Struttura parte calcolo

Come per la parte di grafica, la parte di calcolo viene strutturata allo stesso modo, ovvero con la parte di intestazione e i relativi collegamenti, che saranno poi utili al lavoro della parte grafica, e alla parte di codice centrale per ottenere i risultati di simulazione. Il codice di calcolo, che sará ben trattato nei successivi paragrafi, prevede un sistema di lavoro basato su thread,

ovvero un metodo di esecuzione del codice in modo parallelo, associando ad ogni filone di lavoro una singola telecamera. Nel caso in cui la telecamera sia unica, in esecuzione sará presente una sola thread, la quale lavorerá, vista la struttura di sistema, in modo indipendente e staccato dalla parte di grafica. Il progetto, riassumendo, prevede una doppia suddivisione del lavoro, una sulla parte di interfaccia e l'altra (eventualmente su piu thread e quindi organizzata in modo parallelo) su quella di calcolo e estrapolazione del profilo di liquido.

Lista file di intestazione:

1. *stdafx.h*
2. *targetver.h*
3. *visDLL.h*

Lista file di origine:

1. *dllmain.cpp*
2. *stdafx.cpp*
3. *visDLL.cpp*

3.3 Threads

In questo paragrafo si cerca di dare una infarinatura di base sulla struttura delle threads e sul loro modo di lavorare, oggetto fondamentale del sistema di visione studiato. Una thread si puo definire come una suddivisione di un processo in due o piú filoni, che vengono eseguiti contemporaneamente da uno o piu processori. Una thread é contenuta all'interno di un processo mentre diverse thread contenute nello stesso processo condividono solo alcune risorse (lo spazio d'indirizzamento del processo), mentre processi differenti non condividono le loro risorse. Una thread é composta principalmente da tre elementi: program counter, valori nei registri e stack. Le risorse condivise con le altre threads di uno stesso task sono essenzialmente la sezione di codice, la sezione di dati e le risorse del Sistema Operativo.
La commutazione fra le thread, é proprio questa la caratteristica fondamentale di questa organizzazione, avviene di solito tanto velocemente da dare all'utente l'impressione che tutti i task/processi siano eseguiti contemporaneamente. Nelle architetture multi-processore le thread vengono invece realmente eseguite contemporaneamente, ciascuna su un distinto processore

fisico. Il vantaggio principale nell'utilizzo delle Thread é visibile nelle prestazioni. Operazioni come la creazione, la terminazione e il cambio tra due thread di un processo richiedono sempre meno tempo rispetto alla creazione, terminazione e il cambio di processi. Le thread migliorano quindi anche l'efficienza della comunicazione fra i programmi in esecuzione in quanto tali, all'interno di uno stesso processo, condividono stessa memoria e file, potendo comunicare fra loro senza chiamare l'intervento del sistema opetativo. Viene ora riportata una figura a scopo di esempio della differenza tra una processo eseguito senza l'utilizzo delle thread e lo stesso (a livello di struttura) in un sistema organizzato in modo parallelo.

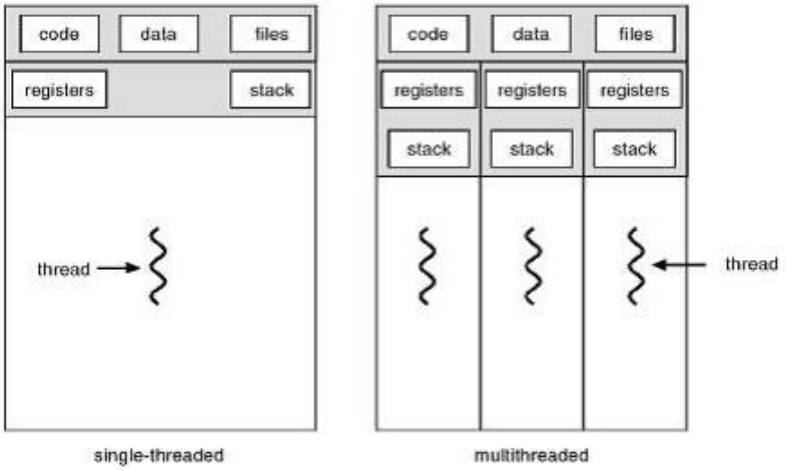


Figura 3.2: Esempio di Threads

3.4 Organizzazione Interfaccia

L'obiettivo di questo paragrafo é quello di spiegare l'interfaccia utente per un corretto funzionamento e utilizzo del software a disposizione. I dettagli sul suo funzionamento interno di codice saranno spiegati successivamente, introducendo ora solo una piccola guida all'utilizzo dello stesso.
L'interfaccia software che viene presentata all'utente durante l'esecuzione risulta la seguente:

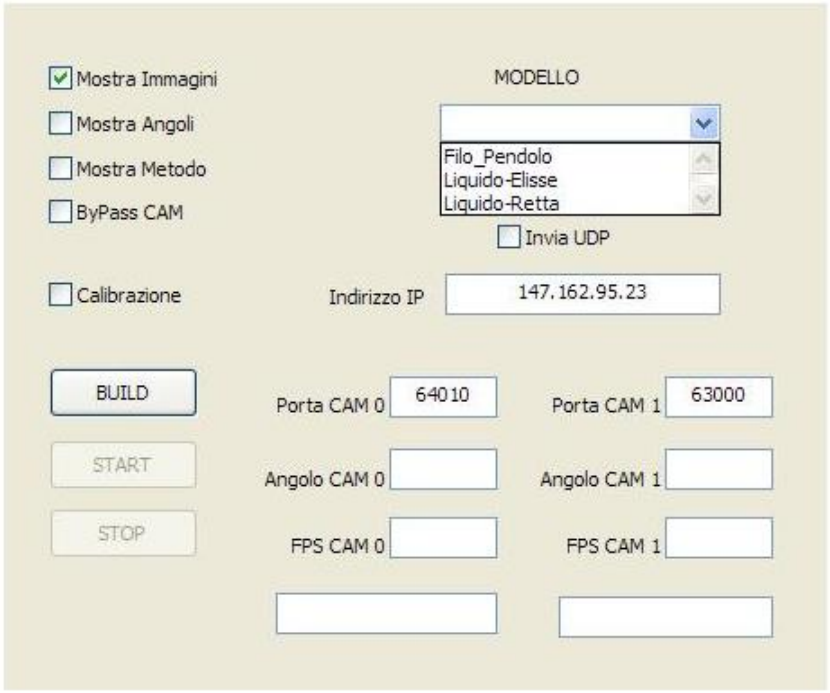


Figura 3.3: Interfaccia disponibile all'utente

In alto a destra dell'interfaccia sará possibile selezionare il modello da utilizzare per la simulazione; tra le scelte possibili si vede la possibilitá di scegliere se utilizzare un modello con pendolo (altra versione giá implementata precedentemente in cui la visione viene utilizzata per ottenere l'angolo di un filo sul quale é appeso un peso) o quello con configurazione con liquido. Per quest'ultima scelta sono possibili due versioni per il calcolo dell'angolo, una tramite interpolazione con retta e una con il metodo dei momenti visto nei primi paragrafi del secondo capitolo. Sempre nella parte a destra sono pre-

senti delle caselle di configurazione per l'invio dei dati (l'angolo ottenuto) ad un controllore/pc identificato dall'indirizzo ip scelto e sulle porte specificate. La versione provvede pure la scelta di non effettuare l'invio dei risultati ottenuti.
Le caselle piu in basso invece mostrano all'utente in tempo reale gli angoli misurati e eventuali informazioni su velocitá di calcolo e sull'identificazione dell'oggetto. La parte a destra risulta doppiata per la presenza di un eventuale seconda telecamera (indispensabile nel modello con filo e pendolo). Per quanto riguarda la parte di sinistra questa definisce, tramite scelta dell'utente, le attivitá da visualizzare e l'avvio/pausa/stop dell'esecuzione. I check box in alto a sinistra mettono a disposizione dell'utente la possibilitá di vedere le immagini acquisite da telecamera in tempo reale (immagini vere o dopo elaborazione), il risultato ottenuto e il metodo utilizzato. Diversi esempi di tali scelte saranno visibili nei successivi capitoli dell'elaborato. La calibrazione del modello, ovvero la scelta del riferimento sul quale poi si baserá il calcolo dell'angolo, avviene tramite apposito pulsante. Gli ultimi tre pulsanti in basso a sinistra vengono utilizzati per mandare in esecuzione il programma, metterlo in pausa, o fermarlo per la scelta di un diverso modello o per la finita simulazione.
Come giá detto, il programma sviluppato, prevede l'integrazione di un vecchio software per il calcolo dell'angolo di un filo al fine di dare un'unica versione di programma sul quale sará possibile tramite scelta, selezione tutte le vecchie versioni e modelli fin qui sviluppati. La trattazione di questa versione non sará prevista nell'elaborato ma ne sará solamente riportato il codice in appendice. Viene ora riportata a scopo illustrativo un'immagine senza elaborazioni successive della visuale della telecamera alla quale successivamente saranno applicati i filtri e le elaborazioni per l'estrapolazione dell'angolo.

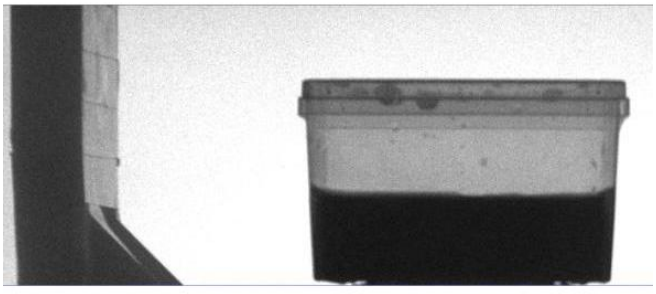


Figura 3.4: Esempio di immagine acquisita

L'immagine proposta non é a colori in quanto le telecamere a disposizione sono in grado di catturare solamente fotogrammi a toni di grigio (sempre con organizzazione in memoria su 3 matrici come nella configurazione RGB vista in primo capitolo, solamente con le tonalitá di grigio su ogni singola matrice).

3.5 Codice di Calcolo

Il paragrafo che segue descrive tutte le procedure e le fasi eseguite durante l'esecuzione del programma per la trattazione delle immagini acquisite da telecamera al fine di calcolare prima il profilo del liquido in movimento, e poi l'angolo di questo. Verrá riportato a spezzoni pure parte del codice prodotto per una piú semplice e chiara spiegazione di questo. Il corpo del codice, per via della sua molteplicitá di modello, si divide tramite operatori if nelle sue tre configurazioni dove ognuna sará associata ad un intero. La classificazione usata nel progetto é:

```
int VERSIONE=NumVer;
/*
0 = PENDOLO CON FILO
1 = LIQUIDO CON ELISSE
2 = LIQUIDO CON RETTA INTERPOLANTE
*/
```

Ovviamente la variabile NumVer sará una delle tante condivise tra la parte di codice che gestisce la grafica e la parte di calcolo puro. A tal fine vengono ora riportate tutte le variabili condivise tra le due parti:

```
typedef struct Data
{
int cam_index;
int area_h;
int area_l;
int livello_bn;
int thickness;
int num_blobs;
double FPS;
int angolo;

DWORD TID;

IplImage* Image;
IplImage* Image_bn;
IplImage* Image_blobs;

char server_IP[BUFFER_SIZE];
unsigned short server_port;
} Data;

extern VISDLL_API bool sending;
extern VISDLL_API bool running;
extern VISDLL_API bool calibration;
extern VISDLL_API bool MostraMetodo;
extern VISDLL_API int NumCam;
extern VISDLL_API int NumVer;
extern VISDLL_API HANDLE* h_Thread;
extern VISDLL_API Data* p_ThreadData;
```

Com'é possibile vedere chiaramente le variabili messe in condivisione e quindi esportate/importate da una parte all'altra sono quelle relative alle informazioni acquisite dall'immagine (l'immagine sará elaborata nella parte di calcolo definita in libreria peró diverse informazioni dovranno essere pure a disposizione dell'utente in fase di esecuzione) contenute nella thread relativa alla singola telecamera. Le ultime variabili invece permettono il cambio in tempo reale di ció che vede l'utente tramite selezione su checkbox e pertanto dovranno essere condivise e aggiornate ad ogni fotogramma elaborato.
Tutte le variabili sono state inserite per comoditá dentro una struttura, ovvero una particolare sezione di memoria in cui tutti i dati vengono raggruppati, per avere un'accesso alla memoria piú comodo nel suo utilizzo.

In seguito viene descritto il funzionamento del corpo principale di calcolo per l'estrazione del profilo del liquido e del relativo angolo di inclinazione. Nel codice completo viene integrata, come giá spiegato, anche la parte di software precedentemente sviluppata per il modello di pendolo che non sará trattata qui nella presente tesi ma solamente allegata nel file completo in appendice. La selezione o l'esclusione di tale codice avviene tramite scelta, durante l'esecuzione del programma da parte dell'utente.

La parte iniziale di codice prevede l'inizializzazione delle variabili e il check delle telecamere (se non vi sono telecamere collegate al pc il programma non andrá in esecuzione). Le variabili piú importanti da inizializzare a inizio programma sono quelle relative alle immagini. Saranno pertanto definite tre immagini dove, in base alla procedura spiegata nei precedenti capitoli, si passerá dall'immagine pura acquisita da dispositivo di visione, successivamente ad un immagine binaria e infine all'immagine finale sulla quale rimarrá presente solamente il profilo del liquido attraverso il quale si potranno effettuare i calcoli per la determinazione dell'angolo di inclinazione di questo. Queste immagini (in memoria sono viste come delle matrici a due o piu dimensioni) sono definite sempre nella struttura riportata prima. Le immagini acquisite avranno dimensione fissa e pari a quella acquisita dalle telecamere.

```
//Inizializzazione camera
Camera.RefreshCameraList();
Camera.SelectCamera(p_Data->cam_index);
Camea.InitCamera();
Camera.StartImageAcquisition();

/*Acquisizione dimensioni frame */
cvWaitKey(500);
unsigned long width,height;
Camera.GetVideoFrameDimensions( &width, &height );

/* Creazione immagini: acquisita, in b/n e con blob (profilo estratto) */
p_Data->Image = cvCreateImage( cvSize(width,height), IPL_DEPTH_8U, CAM_RES_CHANNELS);
p_Data->Image_bn = cvCreateImage( cvSize(width,height),IPL_DEPTH_8U, 1);
p_Data->Image_blobs = cvCreateImage( cvSize(width,height), IPL_DEPTH_8U,
CAM_RES_CHANNELS);
```

Il codice appena inserito sopra riporta l'inizializzazione della telecamera e delle immagini utilizzate.
Dopo la prima fase di definizione delle variabili il tutto viene implementato all'interno di un ciclo while comandato da una variabile corrispondente al pulsante di Start e Stop di interfaccia. In base agli eventi con questi pulsanti sará infatti possibile fermare o far partire il ciclo principale di calcolo che verrá tra poco spiegato.
La prima parte del ciclo inizia con l'acquisizione del frame da telecamera (o piu telecamere in caso di diversa configurazione) e la sua scomposizione in un immagine binaria. L'immagine acquisita, che sará in toni di grigio in quanto la telecamera a disposizione non prevede l'utilizzo a colori, sará poi trasformata in un immagine binaria tramite una sogliatura. Prima di questa procedura peró l'immagine, organizzata in memoria come una a colori anche se in toni di grigio, dovrá essere trasformata in un immagine a una dimensio-

ne. Le immagini a colori infatti, come giá introdotto precedentemente, sono organizzate in memoria come matrici a 3 dimensioni (ognuna per ogni canale di colore) e dovranno pertanto essere scomposte tramite funzione apposita in una versione monodimensionale. Per ulteriori informazioni e dettagli su questa procedura, sia di struttura in memoria che di sogliatura, si rimanda il lettore al primo capitolo di tesi, in cui vengono descritte le operazioni basilari sulle immagini. Il valore di soglia sul quale si andrá a confrontare i vari pixel dell'immagine per determinarne il valore finale 0 o 1, sará impostato dall'utente, tramite una trackbar ovvero una barra a scorrimento in cui sará possibile modificare il valore in un range compreso tra 0 e 255 (0 corrispondente al nero e 255 al bianco). Negli estremi di valore, com'é facile pensare, l'immagine binaria ottenuta sará o completamente bianca o completamente nera. La possibilitá di variare il parametro in questione (parametro che sará poi trasmesso e aggiornato ad ogni ciclo) risulterá molto utile durante l'esecuzione del codice in quanto in base alla quantitá di luce e quindi dall'immagine ottenuta, sará possibile estrarre il liquido (di colore scuro) rispetto allo sfondo (di colore chiaro) andando a pulire ed eliminare eventuali impuritá. Viene ora riportato il segmento di codice relativo a questa prima inizializzazione delle immagini e alla loro scomposizione.

```
//Acquisizione Immagine da telecamera
Camera.AcquireImage();
Camera.getRGB((unsigned char*)(p_Data->Image->imageData),(p_Data->Image->height*p_Data-
>Image->width*3));

//Divisione dell'immagine in matrice monodimensionale
cvSplit( p_Data->Image, p_Data->Image_bn, NULL, NULL, NULL);

//Applicazione della soglia per la binarizzazione dell'immagine
cvThreshold( p_Data->Image_bn, p_Data->Image_bn, p_Data->livello_bn, 255,
CV_THRESH_BINARY );
```

Le tre parti di codice sopra riportate descrivono la fase di acquisizione, trasformazione in matrice monodimensionale e in binaria dell'immagine acquisita.
Vengono ora riportate delle immagini a scopo di esempio delle fasi appena descritte:

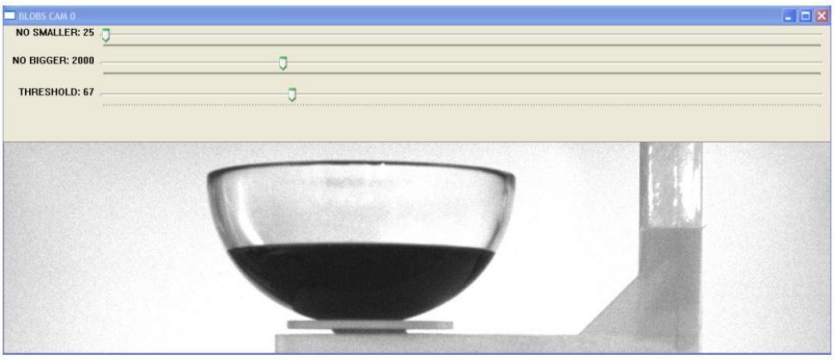


Figura 3.5: Esempio di immagine acquisita

La prima immagine mostra il frame acquisito da telecamera (in tonalitá di grigio come giá prima spiegato). La successiva invece mostra un immagine di esempio del risultato della sogliatura e quindi dell'estrazione del liquido tralasciando lo sfondo e il profilo di supporto del recipiente.

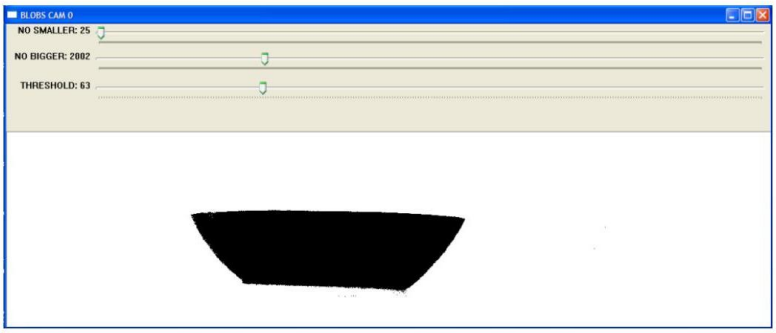


Figura 3.6: Esempio di immagine dopo sogliatura

Le due immagini proposte sono state ottenute con contenitori di liquido diversi in quanto sono stati studiati e provati piú modelli di questi in base ai problemi di visione che saranno trattati in seguito. Il modello finale di simulazione scelto prevede un contenitore quadrato.

Il corpo del codice continua, dopo questa prima fase, con un filtro di pulizia da applicare all'immagine solo nel caso di modello con liquido. La versione del pendolo non sará provvista di tale parte in quando oltre ad essere un aggiornamento introdotto nella nuova versione di software, il caso con filo non presenta tutti i vari problemi di sporco indesiderato nell'immagine (sporco dovuto sia a problemi di luminositá che di supporto del recipienete). La versione con filo infatti non presenta niente altro nel frame acquisito oltre al filo stesso.
La pulizia dell'immagine sará ottenuta, quasi per controsenso, attraverso una procedura di sporcamento della stessa. Il problema che spesso si presenta in questa configurazione consiste nell'avere nell'immagine binaria delle zone, o dei pixel che non sono stati eliminati con la sogliatura, di colore nero. Questi pixel indesiderati, vista la configurazione per il tracciamento del profilo, risultano un problema per la successiva fase di estrazione del pelo del liquido. L'eliminazione di questi punti (rimasti in quanto hanno lo stesso valore di intensitá luminosa del liquido) risulta solitamente abbastanza difficile. In merito a questo problema é stata trovata una soluzione tramite un filtro in grado di sporcare l'immagine binaria andando a ingrandire i pixel singoli neri colorando quelli adiacenti dello stesso colore (nero). In questo modo l'immagine binaria che ne risulta sará la stessa solo con i punti di sporco di dimensione 3-4volte superiore. Questo apparente peggioramento del risultato peró facilita la successiva fase di pulizia in quanto, oltre ad avere meno punti da eliminare (questo comporta una minore onerositá computazione e quindi maggiore performance) si va ad eludere un limite/problema della liberia cvBlob di OpenCV la quale non é in grado di trattare il singolo pixel come un blob, ma ne deve avere almeno due per la sua trattazione.

```
// Filtro per sporcare immagine binaria
cvErode(p_Data->Image_bn,p_Data->Image_bn,NULL,1);

// Analisi immagine per blobs
blobs = CBlobResult(p_Data->Image_bn, NULL, 255);
num_blobs = blobs.GetNumBlobs();

// Metodo di pulizia dei pixel di sporco (ingranditi con funzione cvErode)
double puntox,puntoy;
for(int i=0;i<num_blobs;i++){
      filo = blobs.GetBlob(i);
      area=CBlobGetArea(*filo);
      if(area<300)
            filo->FillBlob(p_Data->Image_bn,CV_RGB(0,0,255));
      if(area==0){
            puntox=CBlobGetXCenter(*filo);
            puntoy=CBlobGetYCenter(*filo);
            (p_Data->Image_bn->imageData+int(puntoy)*step)[int(puntox)]=255;
      }
}
```

La prima parte di codice é relativa alla funzione di sporcatura dell'immagine mentre la seconda ne estrae i blob totali, ovvero quello del liquido in aggiunta a tutti quelli dei pixel singoli sporcati. A questo punto la successiva fase consiste nel eliminare colorando di bianco i blob associati ai pixel indesiderati. Per questo motivo si confronta l'area dei vari blob ottenuti con un valore relativamente piccolo (minore dell'area del liquido in contenitore) e in caso di risultato positivo tali saranno colorati di bianco e quindi parte integrante dello sfondo. Il codice prevede anche il caso in cui, per motivi di errore, non vengano sporcati dei pixel singoli andando ad eliminarli singolarmente. L'aggiunta di tale filtro incrementa la velocitá di esecuzione del codice di calcolo di circa 1Hz ovvero il programa é in grado di calcolare un risultato utile in piú al secondo rispetto allo stesso senza l'aggiunta di questo metodo.

Dopo questa fase di pulizia si arriva al punto centrale del sistema ovvero quello di estrapolare dall'immagine il profilo della superfice del pelo del liquido. L'immagine finora ottenuta rappresenta la massa totale del liquido ma dalla stessa non é possibile estrarre nessuna informazione sul suo angolo di inclinazione. Per questo motivo é stato creato un codice in grado di estrarre la parte superiore della figura del fluido relativa al pelo del liquido. La procedura per fare tale operazione consiste nello scorrere per colonna i vari pixel dell'immagine fin qui ottenuta analizzandone il valore. Avendo un immagine in cui il colore nero risulta solamente associato alla massa del liquido, non appena si incontra un pixel di valore 0 si attua una procedura ulteriore di pulizia. Tutto questo viene realizzato in un ciclo in cui i pixel vengono analizzati secondo il sistema di riferimento visto in capitolo 1. Quando si incontra un valore di pixel nero il codice incrementa l'indice della colonna di un valore fissato e riempie di bianco i successivi fino alla dimensione in altezza massima. In questo modo l'immagine successiva ottenuta sará composta da una figura simile ad un retta ottenuta con i pixel piú esterni/superiori della figura iniziale del liquido. Il parametro di spessore di tale retta (nel caso di quiete ovviamente) viene scelto in tempo reale dall'utente, questo per eludere alcuni problemi visti in esecuzione, sulla continuitá della figura estratta.

Viene riportato ora il codice relativo all'estrazione del pelo del liquido:

```
// Codice per estrazione profilo liquido
unsigned char value;

for(int i=0;i<(int)width;i++){
      for(int j=0;j<(int)height;j++){

            value = *((unsigned char *)(p_Data->Image_bn->imageData + p_Data->Image_bn-
            >widthStep*j)+i);

            if(value==0){
                  for(int k=j+(p_Data->thickness);k<(int)height;k++){
                        *((unsigned char *)(p_Data->Image_bn->imageData + p_Data-
                        >Image_bn->widthStep*k)+i)=255;
                  }
                  j=height;
            }
      }
}
```

La prossima figura illustra un esempio di profilo di superficie di liquido estratto dopo l'esecuzione di questo codice sull'immagine binaria ottenuta:

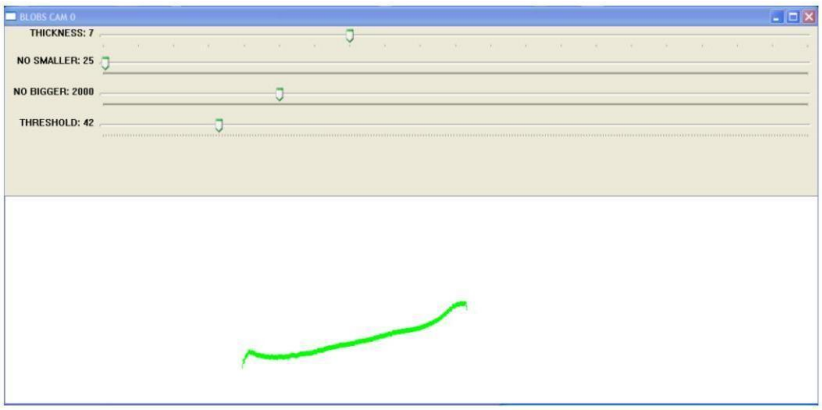


Figura 3.7: Esempio di profilo estratto

L'immagine riportata é stata ottenuta su contenitore quadrato e non con forma semi sferica come le precedenti viste in esempio. Come si puó ora giá meglio immaginare, si puó capire il problema che possono dare dei possibili pixel indesiderati nella parte di immagine appena sopra al liquido. In questo

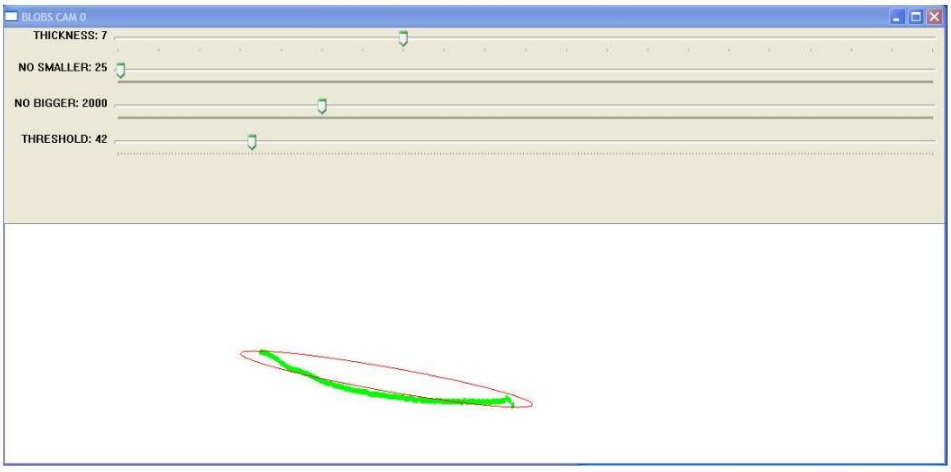
caso il profilo estratto risulterebbe errato con discontinuitá e quindi inutilizzabile al fine del calcolo dell'angolo di inclinazione. In questa finestra di output é presente inoltre una trackbar aggiuntiva (valore Thickness) con la quale si cambia il valore di spessore in pixel del profilo finale estratto.
Il codice riportato sopra fa fede all'ultima versione di software, quella piu performante. A tale proposito saranno discusse le precedenti versioni e le transitorie nei successivi capitoli, trattando i relativi problemi che hanno portato alla loro evoluzione.
A questo punto della trattazione rimane l'ultimo passaggio al fine di raggiungere l'obbiettivo del programma realizzato: il calcolo dell'angolo di inclinazione della figura fin qui trovata. A tale proposito, come giá accennato nella spiegazione dell'interfaccia utente per l'utilizzo del programma, sono stati sviluppati due metodi per calcolare il valore desiderato: uno attraverso l'utilizzo dei momenti e l'altro con intepolazione tramite retta sui punti esterni.

3.5.1 Prima Analisi: Momenti

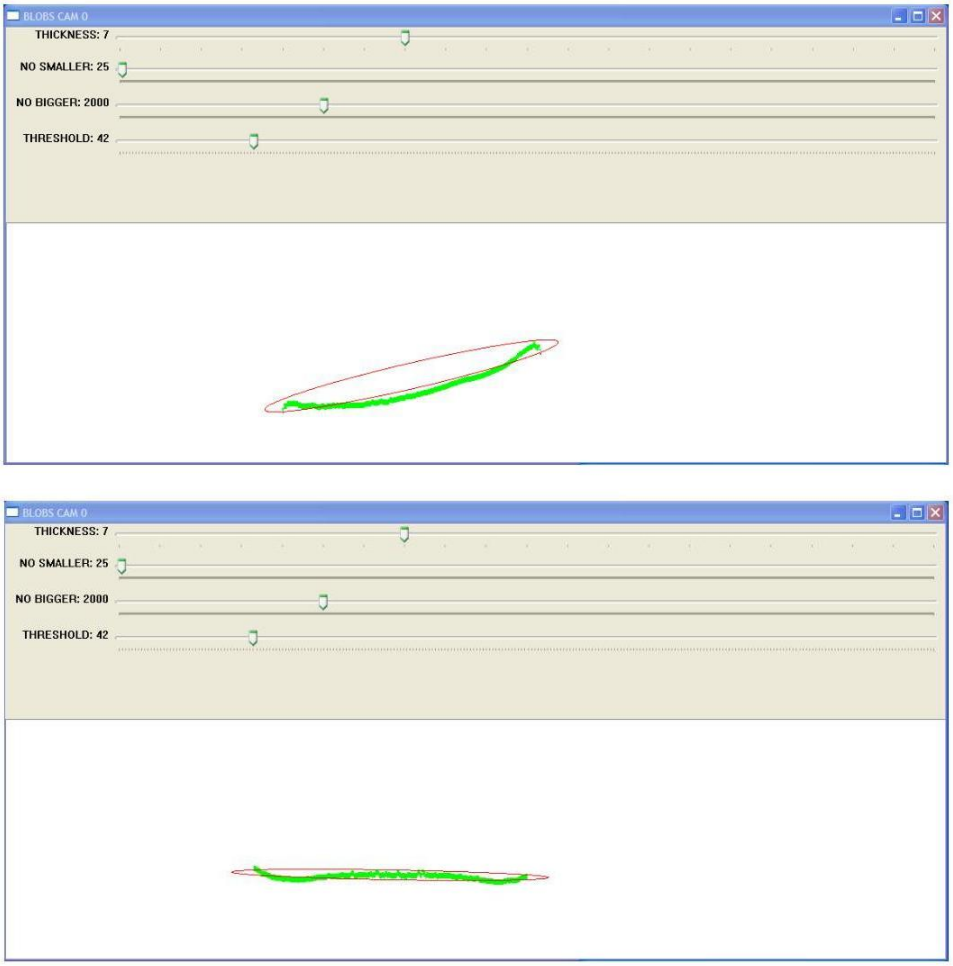
Il primo metodo riportato, e anche il piú performante dei due, é quello del calcolo dell'angolo tramite i momenti di un immagine. Come giá riportato in capitolo 2, questo metodo matematico permette di trovare il valore di inclinazione di una figura geometrica qualsiasi inscritta in un ellisse. In base a questo si andrá a selezione il blob del profilo (e anche l'unico rimasto, visti i passaggi fin qui eseguiti sull'immagine acquisita da telecamera) e tramite metodi in libreria cvBlob si costruirá un ellisse il quale conterrá la figura del blob identificato. L'ellisse cosi ottenuto conterrá al proprio interno (anche se non sempre sará cosi in quanto il metodo si basa anche sui valori centrali del blob e sulla sua massa) la figura del profilo. A questo punto, essendo noti tutti i valori dell'ellisse quali assi, raggi, etcetc. non sará poi cosi difficile trovarne l'angolo di inclinazione rispetto agli assi di riferimento. Viene ora proposto il codice in grado di realizzare le operazioni fin qui descritte:

```
// Filtraggio dei blob per area
blobs = CBlobResult(p_Data->Image_bn, NULL, 255);
blobs.Filter( blobs, B_EXCLUDE, CBlobGetArea, B_LESS, p_Data->area_1 );
num_blobs = blobs.GetNumBlobs();

// Calcolo angolo tramite Momenti
double or=0;
if(num_blobs != 1)
      angolo=angolo_temp;
else{
      filo=blobs.GetBlob(0);
      or=CBlobGetOrientation(*filo);
      angolo=or-180;
      angolo_temp=angolo;
}
```

Nella prima parte di codice viene fatta un ulteriore verifica andando ad escludere tramite funzione tutti i blob con area inferiore ad un valore relativamente piccolo. Questo nel caso in cui per qualche motivo siano rimasti scoperti alcuni pixel o figure alle prime procedure di filtro. La seconda parte invece, oltre al calcolo del valore di inclinazione, viene integrata con una funzione di controllo nel caso in cui in un fotogramma (essenzialmente per motivi di luce e poca visibilitá) non si riesca a trovare un profilo di liquido ben definito. In questo caso é stato previsto come risultato l'invio in output dell'angolo calcolato nel frame precedente vista la grande velocitá di esecuzione del codice e la minima variazione tra un fotogramma e l'altro. Vengono ora riportate delle figure di esempio in cui vengono descritte varie configurazioni di profilo:



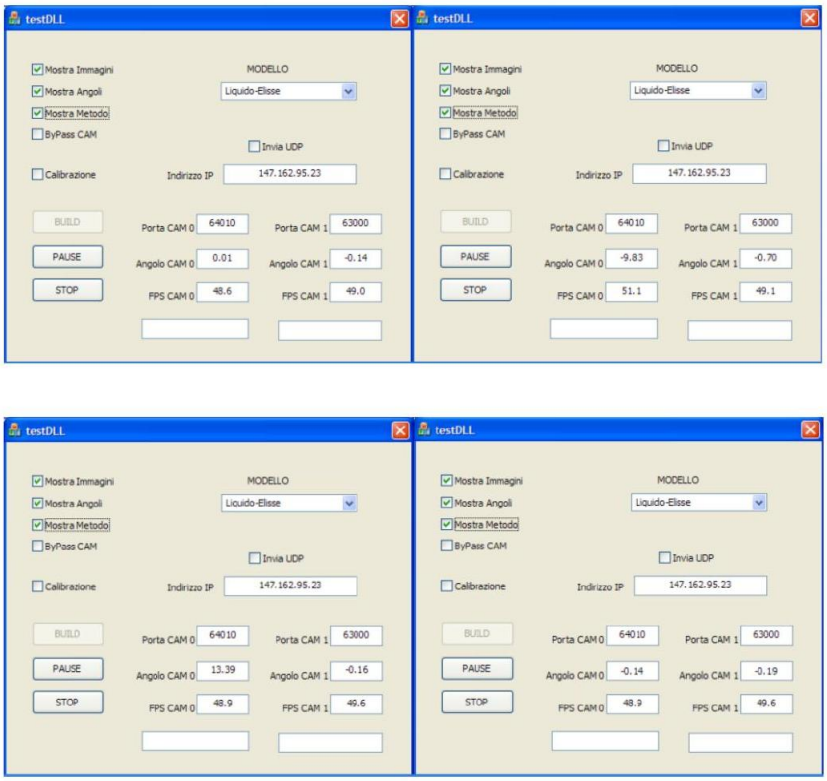
Le due immagini sono state unite per motivi di spazio e rappresentano due configurazioni di angolo di inclinazione, nella prima il liquido risulta in posizione di quiete mentre nella seconda il fotogramma é stato preso in condizioni di turbolenza.



Le successive due invece rappresentano rispettivamente nella prima un'altra condizione di turbolenza ma con angolo opposto alla precedente, mentre la seconda riporta una leggera perturbazione con liquido in leggero movimento superficiale.

Le immagini riportate di esempio mostrano le varie situazioni di angolo con valore positivo, negativo o quasi nullo. In verifica a questi esempi vengono ora proposti i valori ottenuti da queste simulazioni.

Da notare anche la selezione del checkbox Mostra Metodo, attraverso il quale é possibile visualizzare sulle finestre di output anche l'elisse o la retta a seconda del metodo scelto. Deselezionando tale check l'immagine in uscita presenterá solamente il profilo colorato di verde.



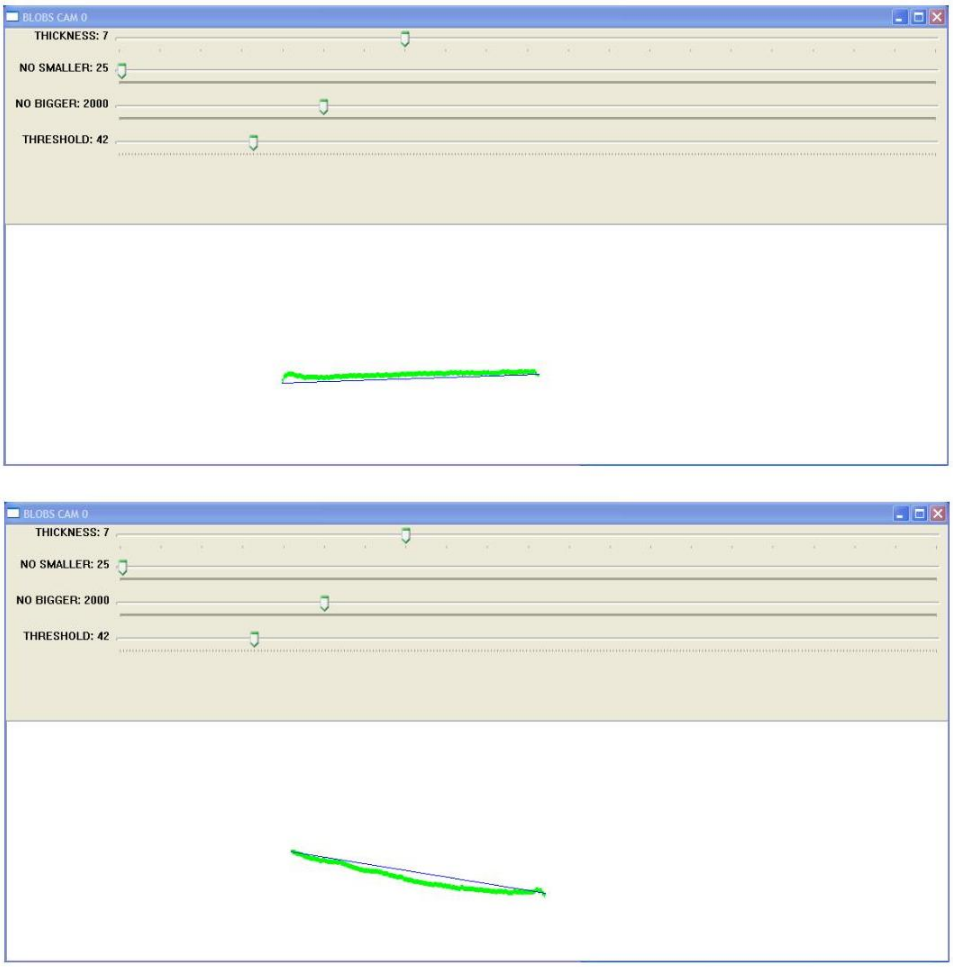
SIMULAZIONE	ANGOLO OTTENUTO
1	**0,01**
2	**-9,83**
3	**13,39**
4	**-0,14**

3.5.2 Seconda Analisi: Retta Interpolante

Il secondo metodo di calcolo dell'angolo prevede come soluzione per il calcolo di questo un approccio ancora matematico ma piu semplice, tramite classica interpolazione. La seconda versione viene sempre applicata all'immagine ottenuta dopo l'estrazione del profilo del liquido e consiste nel tracciare una retta interpolante passante per i punti estremi del profilo. Il codice infatti prende i punti piú esterni a destra e a sinistra del blob e da questi andrá a tracciare una semplice retta di equazione $y = mx + q$ passante per i medesimi. Trovata l'equazione della retta interpolante sará poi facile trovare l'inclinazione di questa matematicamente. Il codice in seguito propone la soluzione appena spiegata:

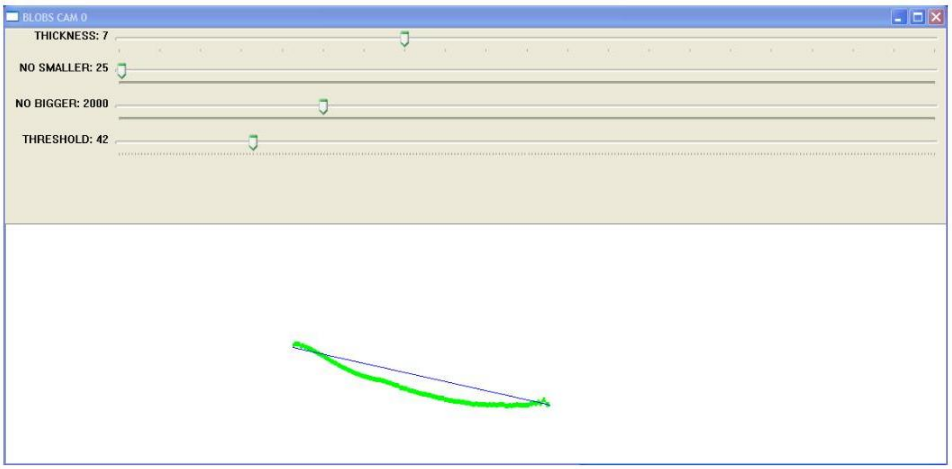
```
// Calcolo angolo tramite Retta interpolante
CvPoint2D64f punto1,punto2;
double ipotenusa;

if(num_blobs==1){
      filo = blobs.GetBlob(0);

      // Calcolo punti esterni
      punto1.x=CBlobGetMinX(*filo);
      punto2.x=CBlobGetMaxX(*filo);
      punto1.y=CBlobGetMaxYatMinX(*filo);
      punto2.y=CBlobGetMinYatMaxX(*filo);

      // Calcolo ipotenusa tramite coefficiente angolare
      ipotenusa=sqrt((punto2.x-punto1.x)*(punto2.x-punto1.x)+(punto1.y-
      punto2.y)*(punto1.y-punto2.y));

      // Calcolo angolo tramite regole sui triangoli
      angolo_retta=asin(fabs((punto1.y-punto2.y))/ipotenusa);
      angolo_retta=angolo_retta*180/3.1415;
}
```

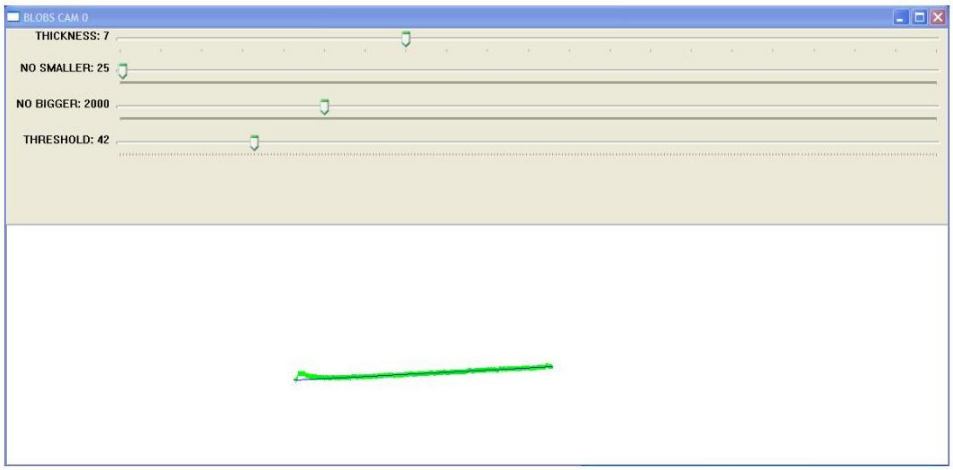
Nel codice riportato é visibile una prima verifica sul numero dei blob seguita dal calcolo vero e proprio dell'angolo tramite regole geometriche sui triangoli. Come per l'altro metodo visto prima vengono ora riportati degli esempi di simulazione con questa configurazione:



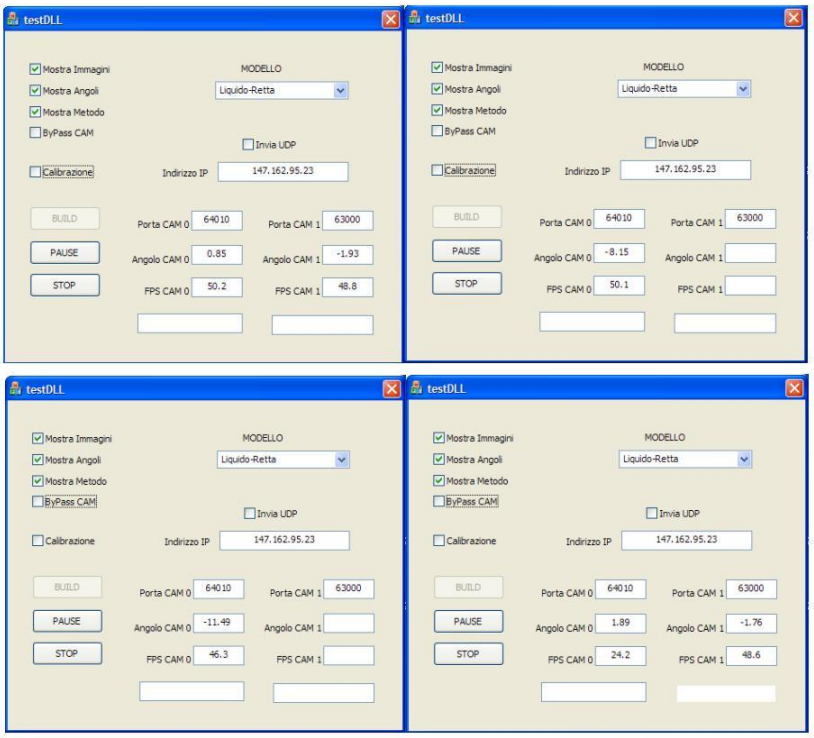
Le due immagini sono anche in questo caso state unite per motivi di spazio e rappresentano due configurazioni di angolo di inclinazione, nella prima il liquido risulta in posizione di quiete mentre nella seconda il fotogramma é stato preso in condizioni di turbolenza (angolo minore di zero).

Da notare anche in questo modello la selezione del checkbox Mostra Metodo, attraverso il quale é possibile visualizzare sulle finestre di output anche la retta di colore blu scuro. Deselezionando tale check l'immagine in uscita sará solamente con il profilo colorato di verde.





Anche per questa configurazione vengono riportati gli angoli misurati degli esempi qui sopra proposti e una medesima tabella di riassunto di questi valori:



SIMULAZIONE	ANGOLO OTTENUTO
1	0,85
2	-8,15
3	-11,49
4	1,89

Anche in questa configurazione é stata attivata la visualizzazione del metodo scelto, per questo motivo sono presenti le rette di colore blu in aggiunta sulla finestra di uscita. Questo metodo, anche se semplice di configurazione e di idea riulta molto piú instabile rispetto al precedente per i classici problemi che circondano la visione e le sue applicazioni. Questo approfondimento sará trattato piú nel dettaglio neli capitolo finale della tesi.

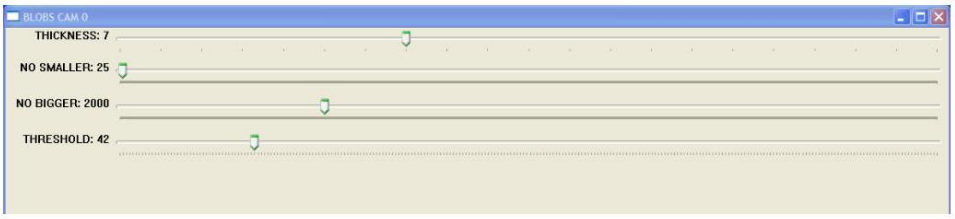
Capitolo 4

ANALISI E RISULTATI

Il seguente capitolo prevede un analisi delle simulazioni e dei risultati ottenuti dall'utilizzo del software introdotto nel precedente capitolo. Verrá trattata un'iniziale spiegazione sulla configurazione dei parametri per un corretto utilizzo del codice e successivamente alcuni risultati ottenuti dalle varie prove e dalle configurazioni di queste. Verranno riportati anche dei modelli di studio e dei confronti, per capire la bontá del software, con i dati reali.

4.1 Configurazione Software

Nel primo paragrafo si da una delucidazione sull'utilizzo del software per quanto riguarda la sua configurazione dei parametri attraverso i trackbar (particolari barre che appariranno all'utente non appena si fa partire il programma). Attraverso questi 4 parametri modificabili sará possibile variare le variabili associate (questo in tempo reale) e ottenere risultati e prestazioni diverse.

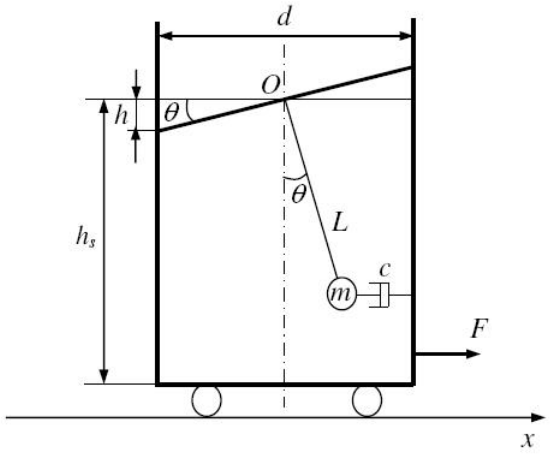


La prima barra in alto é stata inserita in aggiunta alla vecchia versione del software per il modello con il pendolo e pertanto il suo variare dará risultato

utile solamente nella versione aggiornata con modelli di liquido selezionati. Con la barra citata ovvero Thickness sará possibile modificare il valore dello spessore del profilo del liquido ottenuto dall'ultima fase di calcolo vista nel precedente capitolo. Con questo parametro (in pixel come valore) si elimina pure un possibile problema di discontinuitá della figura nel caso di grandi oscillazioni del fluido comportando quindi una perdita del blob e del relativo angolo di inclinazione che si dovrá poco dopo calcolare.
Le successive due barre ovvero le No smaller e No bigger vengono utilizzate indifferentemente dalla versione e dal modello scelto. La funzione delle due é quella di eliminare o meno i blob trovati tramite analisi sull'immagine di dimensione maggiore o minore di quella specificata. Anche in questo caso il valore modificato viene misurato in pixel.
L'ultima barra invece, forse la piú importante, tratta il parametro che digitalizza l'immagine acquisita da telecamera e viene modificata dall'utente in modo da eliminare lo sfondo e gli oggetti indesiderati tramite la sogliatura. Ad ogni esecuzione del programma infatti risulta utile viste le differenti condizioni di lavoro e di luce in ambiente, andare a modificare tale parametro per permettere al codice di calcolo di lavorare con una figura appropriata ed esatta della visione reale.
Attraverso le quattro barre fin qui spiegate é possibile modificare le varie condizioni di lavoro e le immagini sulle quali poi il codice andrá a lavorare. Un corretto setting in questa configurazione comporta risultati nettamente migliori e coerenti di angolo di inclinazione.

4.2 Modelli di contenitore

Nel seguente paragrafo vengono trattati e spiegati brevemente i modelli di recipiente (questi si possono associare a dei pendoli semplici) al fine di uno studio di questi per un successivo confronto con i dati reali acquisiti dal programma. Nel successivo paragrafo infatti verranno trattate delle analisi con i dati acquisiti da software e i modelli studiati su carta.
Nel caso di movimento su un percorso orizzontale rettilineo, senza accelerazioni improvvise o repentini cambi di direzione tali da determinare un moto troppo turbolento, le oscillazioni di un liquido in un contenitore cilindrico possono essere approssimate come un fenomeno bidimensionale, nel quale é dominante il primo modo di oscillare. Le oscillazioni all'interno del contenitore possono essere studiate utilizzando il modello del pendolo in figura dove la superficie del liquido é considerata come un piano perpendicolare al pendolo e passante per O.



L'angolo di oscillazione del pendolo θ rappresenta l'entitá dell'oscillazione del liquido, cioé l'angolo della sua superficie dall'orizzontale; h é lo spostamento della superficie del liquido dal suo livello statico h_s e si ottiene da:

$$h = \frac{d}{2} \cdot \tan\theta$$

dove d é il diametro del contenitore cilindrico.
Questo modello contiene uno smorzatore che rappresenta un coefficiente di viscositá equivalente c, che dipende dalla viscositá del liquido e dall'attrito tra esso e la parete del contenitore; inoltre é presente una massa concentrata m, rappresentativa della quantitá di liquido; L é la lunghezza equivalente del pendolo. I parametri L e c sono solitamente determinati sperimentalmente ma i loro valori possano essere ricavati anche a partire da alcune relazioni matematiche.

L'equazione del moto del modello considerato risulta in questo caso:

$$mL^2\ddot{\theta}(t) + cL^2\dot{\theta}(t)\cos^2\theta(t) + mgL\sin\theta(t) = -mL\ddot{x}(t)\cos\theta(t)$$

Sotto le ipotesi che il pendolo si muova per piccole oscillazioni dall'asse verticale, si possono introdurre le seguenti approssimazioni:

$$\sin\theta = \theta$$

$$\cos\theta = 1$$

$$\dot{\theta}^2\theta = 0$$

Ottenendo cosí le equazioni linearizzate:

$$mL^2\ddot{\theta}(t) + cL^2\dot{\theta}(t) + mgL\theta(t) = -mL\ddot{x}(t)$$

Senza riportare tutti i vari passaggi e teorie sul calcolo della frequenza naturale dominante di un liquido in contenitori di varie geometrie, viene ora riportata equazione per il calcolo di questa in caso di modello con contenitore cilindrico:

$$f_n = \frac{1}{2\pi}\sqrt{\frac{g}{R} \cdot \epsilon \cdot \tanh\frac{\epsilon \cdot h_s}{R}}$$

dove si é indicato con h_s il livello statico del liquido, R é il raggio del contenitore, g l'accelerazione gravitazionale e ϵ é la radice della derivata prima della funzione di Bessel del del primo tipo pari al valore 1.841. Questa frequenza naturale si mantiene per qualsiasi direzione in cui é mosso il contenitore.
La lunghezza equivalente del pendolo L si puó ricavare analiticamente a partire dalla frequenza naturale. Dividendo i termini dell'equazione del moto per mL^2 si ottiene:

$$L = \frac{g}{(2\pi f_n)^2}$$

Per il calcolo dello smorzamento invece si utilizza la relazione di Mikishev e Dorozhkin che propongono la seguente equazione, proveniente da test sperimentali, che per un contenitore cilindrico puó essere utilizzata per calcolare con buona approssimazione il fattore di smorzamento:

$$\xi = 0.79 \cdot \sqrt{\gamma}\left[1 + \frac{0.318}{\sinh(1.84 \cdot h/R)} \cdot (1 + \frac{1 - h/R}{\cosh(1.84 \cdot h/R})\right]$$

dove:

$$\gamma = \frac{v}{\sqrt{gl^3}}$$

e v pari alla viscositá del liquido.
Per $h > 2R$ la relazione si riduce a:

$$\xi = 0.79 \cdot \sqrt{\gamma}$$

4.3 Analisi risposta ad impulso

In questo paragrafo viene proposta un'analisi del sistema considerato per un confronto tra i dati ottenuti tramite modello su carta (in particolare sul valore di smorzamento e frequenza naturale del liquido nel contenitore usato) con i valori di angolo acquisiti da programma nel caso in cui il sistema venga eccitato da un impulso. Con le trattazioni del precedente paragrafo si é in grado di stimare un modello equivalente di pendolo con una sua frequenza naturale e uno smorzamento.

4.3.1 Risposta su contenitore cilindrico

Con le formule viste, applicate su un semplice codice in matlab, il modello studiato riporta i seguenti valori:

$$\xi = 0,098$$

$$f_n = 22,8801 Hz$$

Per una verifica dei valori di smorzamento e di frequenza verrá utilizzato il metodo del decremento logaritmico in una risposta del sistema ad impulso. Per emulare questo tipo di situazione, vista l'impossibilitá di applicare un impulso iniziale, si é utilizzata una tecnica diversa. Si é applicato infatti un moto uniforme in velocitá con un improvviso arresto di questo. Tale situazione, nell'ultima parte di movimento, imita l'azione di un impulso in quanto il liquido, in caso di arresto improvviso comincerá ad oscillare e a fermarsi con un graduale smorzamento. Attraverso il sistema di visione implementato e spiegato precedentemente si andranno a prelevare i valori dell'angolo di oscillazione sui quali poi, applicando il metodo del decremento logaritmico, si andrá a stimare lo smorzamento del liquido e la frequenza naturale per un confronto. Viene ora riportato in figura l'andamento dell'angolo di oscillazione del liquido nel caso di contenitore cilindrico:

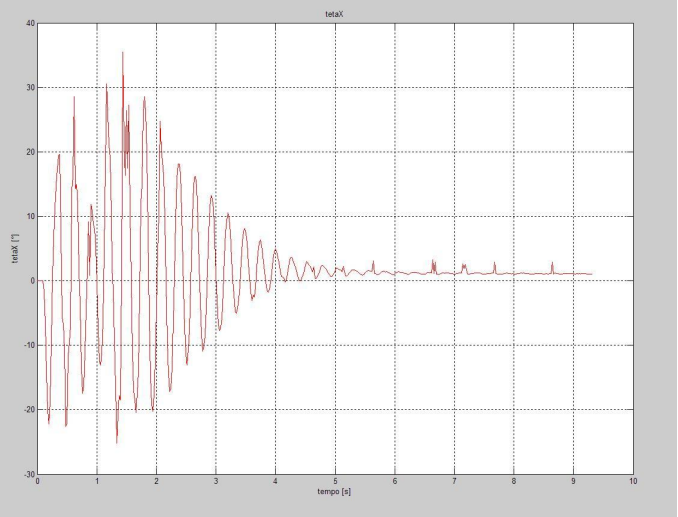
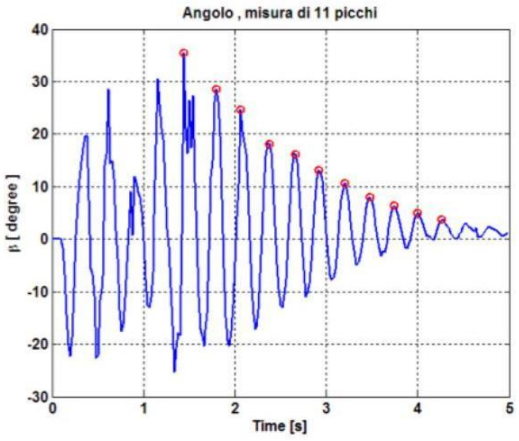


Figura 4.1: Risposta impulsiva su contenitore cilindrico

Come si puó ben vedere in figura l'andamento iniziale di oscillazione, parecchio strano e senza senso, é causato dalla partenza del supporto in movimento rettilineo mentre dopo lo stop, ovvero dal secondo 1,5 il liquido comincerá a smorzarsi fino a raggiungere una posizione di quiete. Su questo andamento finale di angolo sará applicato il metodo del decremento logaritmico che andrá a prelevare i picchi di smorzamento per andare a stimare la frequenza natutale del sistema in base ai dati sperimentali. Vengono ora riportate due figure relative a questa procedura e il valore di risultato ottenuto.



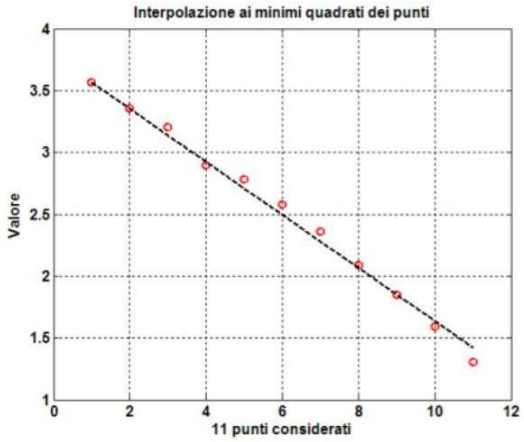


Figura 4.2: Applicazione del metodo del decremento logaritmico su contenitore cilindrico

Il valore della frequenza di risonanza ottenuta con questo metodo sará pari a 22.5370 Hz. Come si puó ben vedere i due valori hanno un errore tra di loro molto piccolo in fede alla bontá della lettura della visione. Attraverso questo confronto si capisce la linearitá del sistema e pure la velocitá di esecuzione del software appropriata. L'andamento dell'angolo infatti risulta molto fluido e con quasi nessun punto di discontinuitá. A tal fine il metodo del decremento logaritmico é in grado di operare sui picchi dell'oscillazione e proporre un risultato ben definito. In caso di sistema di visione poco funzionante e/o sistema non lineare l'andamento dello smorzamento sarebbe stato piú confuso e irregolare.

4.3.2 Risposta su contenitore quadrato

La stessa procedura é stata effettuata anche su un modello di recipiente a forma quadrata applicandone lo stesso metodo e lo stesso confronto. A questo proposito viene ora riportata una figura della risposta ad impulso su questa configurazione:

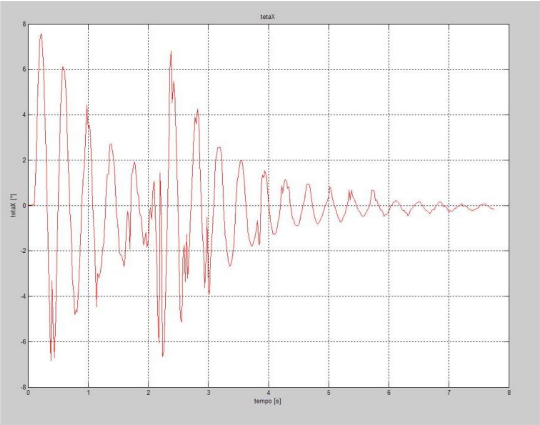
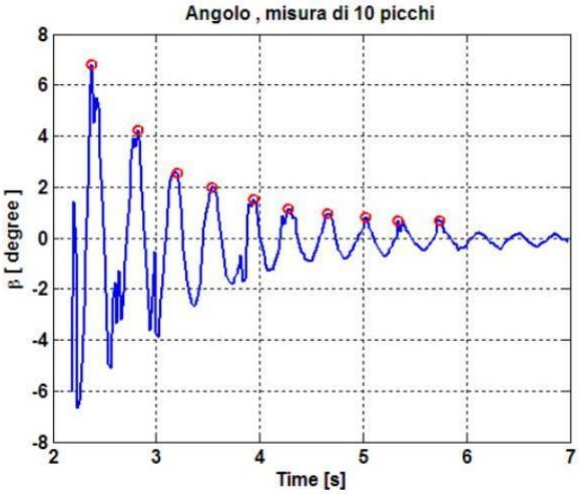


Figura 4.3: Risposta impulsiva su contenitore quadrato

Allo stesso andamento di oscillazione viene ora applicato il metodo del decremento logaritmico per il calcolo della frequenza di risonanza, la procedura risulta identica a quella applicata per il caso di recipiente cilindrico:



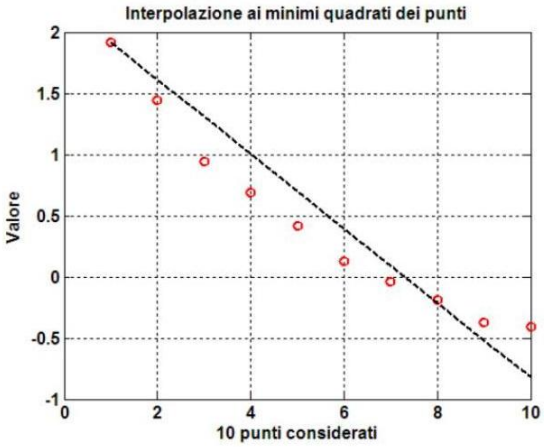


Figura 4.4: Applicazione del metodo del decremento logaritmico su contenitore quadrato

Anche in questo caso il metodo ha ottenuto un valore di frequenza naturale prossimo a quello calcolato su carta tramite modello di paragrafo precedente. In questo primo caso infatti il valore della frequenza naturale f_n si calcola tramite:

$$w_n^2 = \pi(2n-1) \cdot \frac{2g}{h_s} \tanh\left[\pi(2n-1)\frac{2h}{h_s}\right]$$

dove n rappresenta l'ordine del modo di vibrare (pari ad 1 nel caso studiato e a frequenza piú bassa).
Tramite questa formulazione si calcola un valore di frequenza di risonanza pari a 17.16Hz invece con il secondo metodo, quello del decremento, il valore ottenuto é pari a 16.9992 Hz.

Anche in questa configurazione di sistema i risultati ottenuti nei due modi sono quasi prossimi tra loro indicando ancora una volta la sufficiente velocitá di calcolo dell'algoritmo. Per questo motivo l'andamento di angolo analizzato produce risultati che rispecchiano quelli teorici. Se cosi non fosse si noterebbero valori di angoli strani e uno smorzamento non lineare, indice di una possibile poca velocitá di esecuzione o di un non corretto calcolo di inclinazione del liquido.

4.4 Studio della movimentazione per versamento di un liquido

Il paragrafo che segue propone uno studio sulla movimentazione dell'organo terminale del robot Adept Four a disposizione per far versare il liquido contenuto nel recipiente in modo esatto dentro un ipotetico contenitore posto al di sotto del supporto. Prima di iniziare con la trattazione vera e propria viene riportata una foto dell'organo terminale per semplificarne la comprensione di struttura e funzionamento:

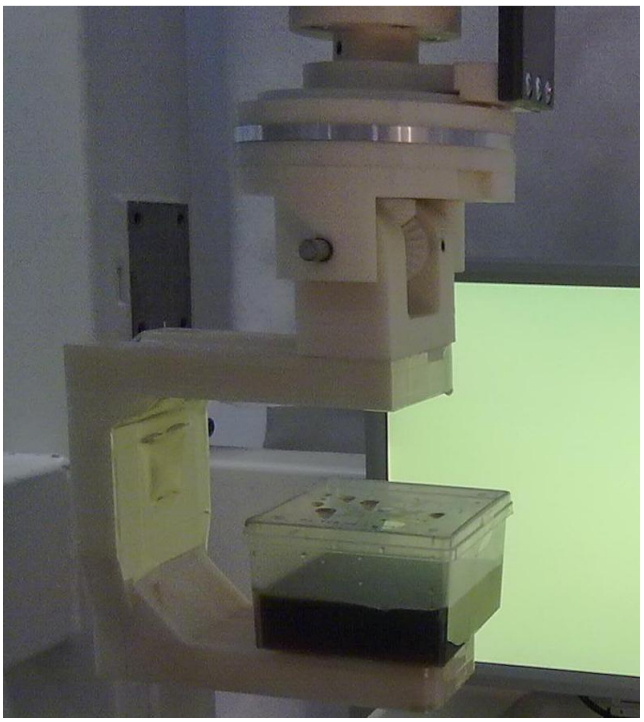


Figura 4.5: Supporto a C utilizzato con contenitore quadrato

La seconda rappresentazione riporta il supporto stilizzato,rappresentazione che nelle seguenti trattazioni e esempi verrá utilizzata per permettere una piú chiara comprensione:

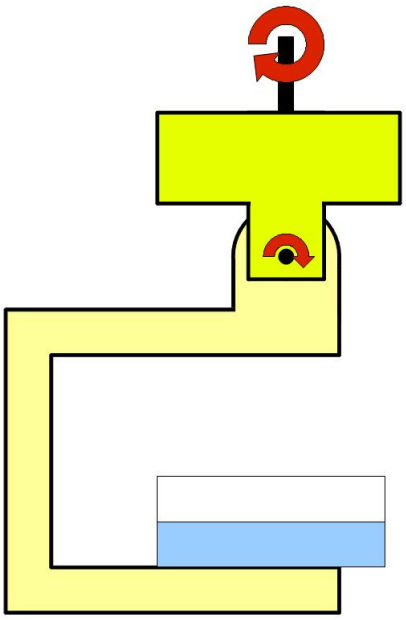


Figura 4.6: Supporto a C stilizzato

Come si puó vedere il supporto mette a disposizione un piano di appoggio sul quale fissare il tipo di contenitore. Attraverso l'accoppiamento in alto é possibile trasformare la rotazione rispetto all'asse z del manipolatore in una rotazione rispetto all'asse ortogonale inserito. In questo modo é possibile far ruotare il supporto finale al fine di compiere un movimento simile all'azione di versamento di un liquido (o al contrario, se si vuole farlo rimanere dentro in caso di disturbi e vibrazioni, di controllo di questo).

4.4.1 Soluzione tramite semplice rotazione

Una prima soluzione al problema posto sarebbe quella di far ruotare semplicemente (considerando sempre il caso con manipolatore in condizioni statiche) il supporto ma tale movimentazione porterebbe alla dispersione del liquido su tutta la superficie del piano di lavoro. Questa prima soluzione errata viene riportata nella rappresentazione del modello seguente al fine di spiegarne bene pure il funzionamento:

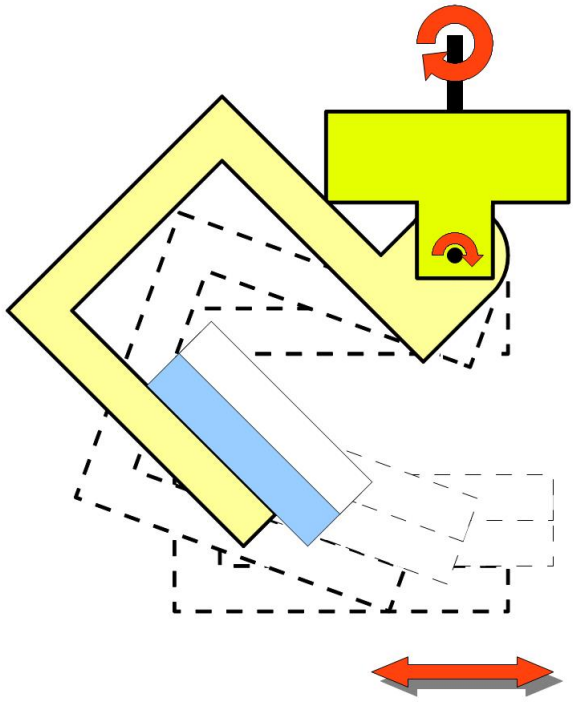


Figura 4.7: Soluzione con rotazione

Con questa movimentazione il liquido verrebbe versato in una gran parte del piano di lavoro, nel caso in figura la zona interessata sarebbe quella delimitata dalla freccia rossa al di sotto del supporto. Questa soluzione risulta sicuramente errata.

4.4.2 Soluzione con rotazione e traslazione

Una alternativa ben piú efficacie di questa, consiste nel far traslare, oltre alla rotazione appena vista, il supporto secondo delle precise regole. Il metodo si basa sulla scelta di un punto fisso attorno al quale il resto dovrá ruotare, attorno ad una circonferenza. Il punto fisso sul quale si va a ruotare puó essere scelto a piacere ma nel caso del versamento del liquido questo sará scelto sull'estremitá destra del contenitore del liquido. In questo caso tutto il resto ruoterá attorno al quel punto preciso e il liquido sará versato in modo corretto in un ipotetico contenitore posto sotto il supporto. La circonferenza che dovrá seguire il supporto sará definita dal punto fisso e dal punto sull'asse

di rotazione del supporto stesso. La distanza tra questi due punti determinerá un raggio sulla quale sará possibile costruire la circonferenza di percorso. La figura sotto riporta la spiegazione fin qui fatta dal punto di vista grafico:

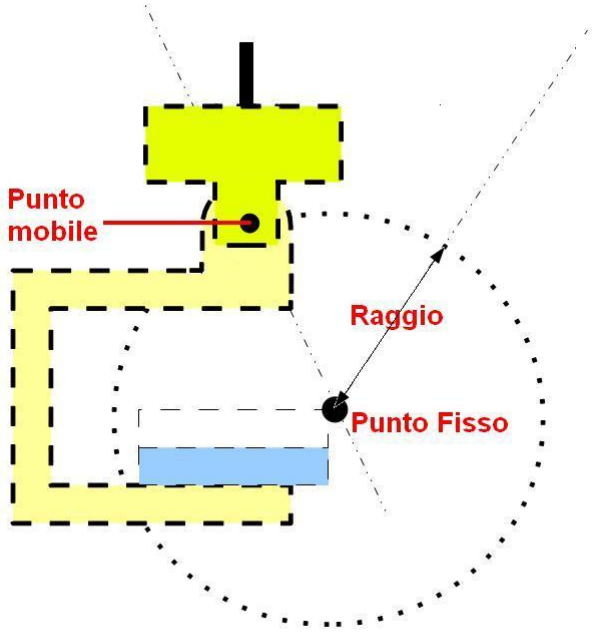


Figura 4.8: Determinazione dei punti

A questo punto, scelto un angolo di rotazione, il supporto dovrá muoversi sul perimetro della circonferenza (in particolare il punto mobile fissato sull'asse di rotazione) su un arco costruito in base al medesimo angolo. Durante questa traslazione il supporto dovrá ruotare in modo proporzionale allo spostamento in modo tale da completare l'angolo di rotazione in corrispondenza dell'arrivo del punto a fine arco.
A tal merito vengono riportate ora delle figure in sequenza nel caso di un singolo spostamento su arco di circonferenza:

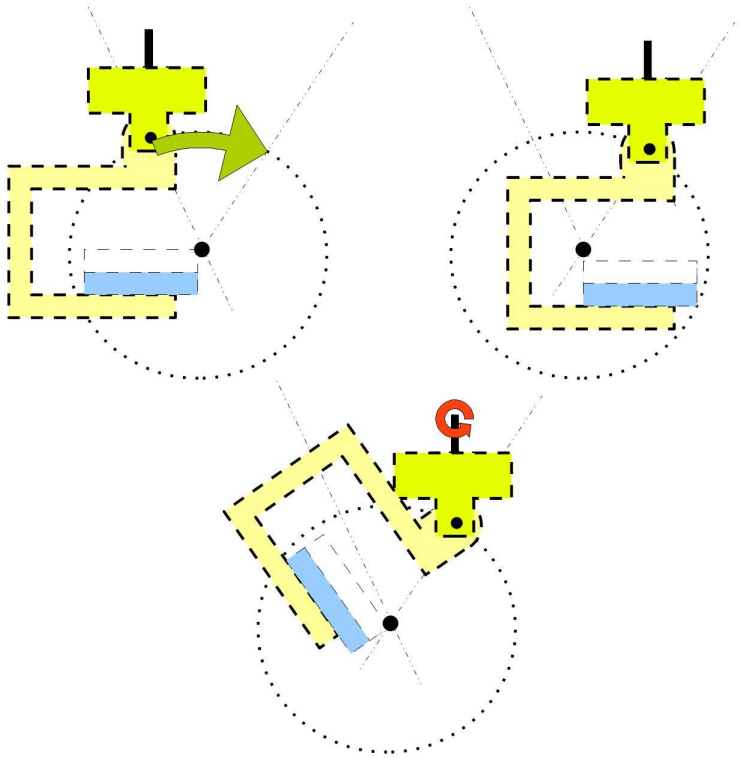


Figura 4.9: Esempio di rotazione e traslazione

Ovviamente le due componenti, di rotazione e traslazione, dovranno essere eseguite contemporaneamente durante lo spostamento in modo da mantenere sempre fisso il punto scelto attorno al quale ruotare, e non in due parti come riportato in figura. Se l'andamento fosse diviso, la soluzione non sarebbe poi tanto diversa da quella proposta prima.

Il linguaggio di programmazione utilizzato sul robot Adept Four (Adept V+) non prevede delle funzioni di spostamento curvilineo pertanto il seguire del perimetro della circonferenza dovrá essere simulato con brevi tratti rettilinei. Durante questo singolo tratto, pari ad un grado o piú, a scelta dell'utente, si dovrá verificare la rotazione prevista e la traslazione determinata dalla proiezioni ortogonali del vettore di spostamento. Tramite seno/coseno e angolo si andrá a determinare la matrice di roto-traslazione per effettuare il singolo intervallo. In modo ciclico poi saranno effettuati tutti gli spostamenti fino a raggiungere il punto finale, mantenendo fisso quello di centro di circonferenza durante la movimentazione.

In esempio viene riportato ora lo spostamento su piú intervalli rettilinei per raggiungere la rotazione voluta attorno al punto fisso:

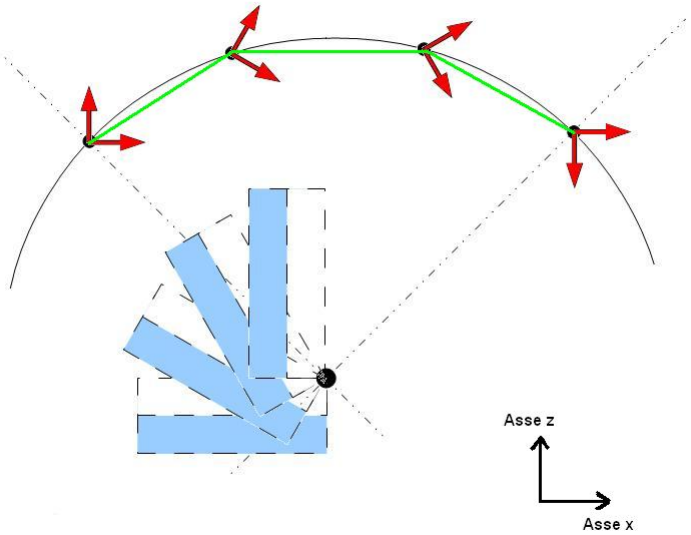


Figura 4.10: Sistemi di riferimento su piu intervalli

La figura riportata sopra considera un esempio con rotazione di 90 gradi attorno al punto fisso, anche in questo caso scelto sull'estremitá del contenitore per permettere al liquido di essere versato correttamente. Il tratto di circonferenza é stato diviso, per semplicitá, in 3 andamenti rettilinei e quindi con una rotazione (sempre attorno all'asse z del manipolatore vista la configurazione e il funzionamento del supporto a C) pari a 30gradi per ogni singolo intervallo. Durante ogni spostamento avverrá quindi anche la rotazione per mantenere il punto del centro di circonferenza sempre fisso. Come risulta semplice pensare, piú é elevato il numero di intervalli per eseguire l'angolo di rotazione, piú l'andamento nelle movimentazioni sará fluido e piú sará immobile il punto del centro di circonferenza. L'utente avrá quindi la possibilitá di scegliere il numero di intervalli tramite modifica di un ciclo. Il programma richiede l'inserimento della distanza in x e in z tra i due punti (quello fisso e quello mobile posizionato sull'asse di rotazione del supporto) e l'angolo di rotazione. Il punto fisso puó essere scelto a piacere in tutto lo spazio di lavoro.

In aggiunta a questa soluzione é possibile integrare il tutto con una traslazione in avanti o indietro sull'asse x. Per questa semplice soluzione basta moltiplicare la matrice di roto-traslazione che identifica lo spostamento in ogni singolo tratto con una matrice di traslazione rispetto all'asse x, con valore positivo per aggiungere uno spostamento verso destra o con valore negativo con spostamento a sinistra.
Tutte queste trattazioni vengono riportate in appendice dove si puó consultare il codice completo in V+ fin qui descritto.

Capitolo 5

CONCLUSIONI E SVILUPPI FUTURI

L'ultimo capitolo della tesi propone un'analisi dei risultati ottenuti e dei problemi incontrati (risolti e non) riguardanti il progetto sviluppato. A tal proposito vengono anche discussi i possibili sviluppi futuri e le possibili migliorie da applicare al sistema al fine di ottenere migliori prestazioni.

5.1 Sviluppi Futuri

5.1.1 Sistema DRC

In questo paragrafo si studia l'integrazione del sistema di visione implementato con il sistema di controllo DRC. Il controllo prevede giá un sistema di visione ma utilizzabile solamente nel caso di carico sospeso in cui il calcolo dell'angolo di inclinazione sará applicato ad un filo.

Il sistema di controllo DRC (Delayed Reference Control) é un sistema di controllo che prevede una strategia basata sul controllo attivo delle vibrazioni con simultaneo monitoraggio del percorso in esecuzione su sistemi lineari a piú gradi di libertá. Il sistema permette di ridurre le vibrazioni elastiche garantendo un movimento coordinato tra un carico e il manipolatore. Il DRC viene applicato per smorzare l'oscillazione di un carico sospeso alla piattaforma mobile Adept Four , particolare robot parallelo, per mezzo di un cavo inestensibile. Il modello si compone quindi di un pendolo fissato all'organo terminale del robot parallelo monitorato attraverso un sistema di visione (giá presente e funzionante) in grado di fornire l'oscillazione del peso al controllore, che fornirá un opportuna compensazione per smorzare le oscillazioni

vista la non rigidezza del carico applicato. Viene ora descritto brevemente la configurazione del sistema e il suo set up sperimentale:

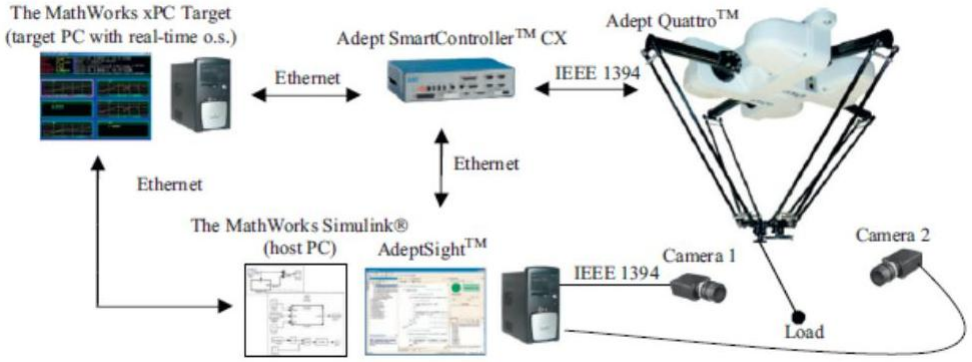


Figura 5.1: Sistema di controllo DRC

In sequenza vengono ora elencati i componenti e le caratteristiche del sistema descritto in Figura 5.1.

Il set-up é composto da:

- **Telecamere FireWire:** Le telecamere FireWire sono basate sui chip CCD o CMOS attraverso i quali vengono trasformate le componenti di luce incidente in elettroni (le telecamere trattano solamente dei segnali elettrici). La loro area fotosensibile e anche i singoli pixel sono di dimensioni estremamente contenute. Se in un pixel si accumula una quantitá notevole di fotoni, allora si avrá la generazione di una tensione elevata. Se si tratta invece di pochi fotoni, la tensione rimarrá bassa. La tensione,essendo un valore analogico, nella seconda fase della digitalizzazione, un convertitore A/D la trasformerá in un valore digitale, rendendo disponibile l'immagine digitale grezza.
  Le telecamere FireWire possono scambiare dati con ogni altro dispositivo FireWire nel caso entrambi i dispositivi usino lo stesso protocollo. I dati che possono essere scambiati possono essere diversi, a partire da dati d'immagini e audio fino ai parametri per la regolazione della fotocamera, videocamera o telecamera stessa.

- **Computer:** Il computer viene utilizzato per diverse funzioni, le principali consistono nell'esecuzione del programma di visione che dovrá

trasmettere in tempo reale l'angolo di inclinazione misurato, la gestione del sistema DRC tramite schema simulink che verrá poi caricato da un PC host sul PC di destinazione dove si gestirá il segnale di controllo verso il controllore Adept. Verranno quindi eseguite tre attivitá concorrenti in tempo reale al fine di gestire il movimento del robot tramite controllore, l' interpretazione dei dati provenienti dal sistema di visione e la pianificazione della traiettoria di percorso.

- **Manipolatore parallelo Adept Quattro:**

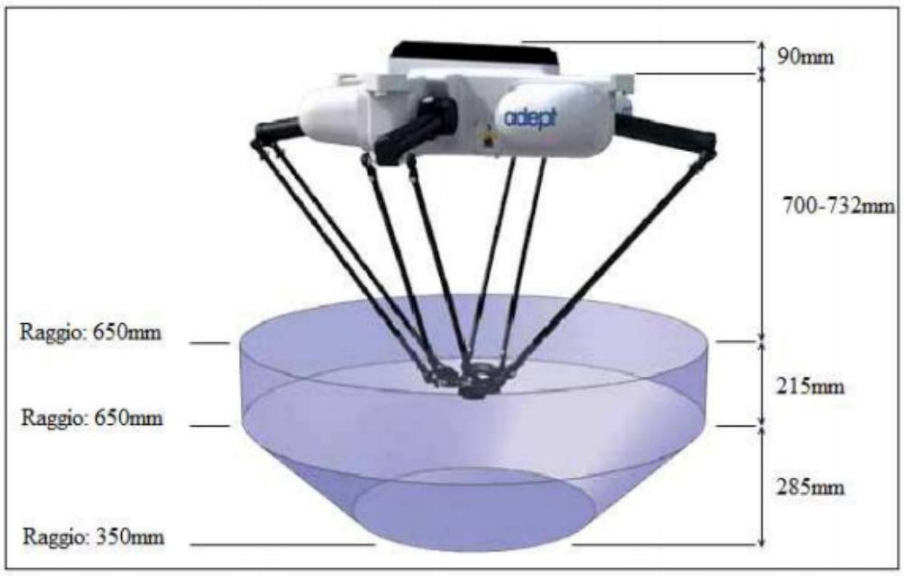


Figura 5.2: Robot Parallelo Adept Four

Il manipolatore di tipo parallelo presenta quattro bracci identici ciascuno costituito da due link: uno piú vicino al telaio chiamato braccio superiore o link 1, l'altro definito braccio inferiore o link 2. Il collegamento a telaio del manipolatore, come spesso capita per le macchine parallele, avviene sulla parte superiore, dove sono presenti quattro attuatori, protetti da calotte in alluminio. Questo robot, prodotto dall'Adept Technology nasce in ambito accademico e rappresenta l'ultima evoluzione della famiglia dei DELTA-Robot, con la peculiaritá di dotare l'end-effector di quattro gradi di libertá, per la realizzazione di moti a elevate velocitá e accelerazioni. La particolare struttura della piat-

taforma mobile garantisce la rotazione dell'organo terminale attorno al suo asse verticale. L'Adept Quattro si presta a movimentazioni veloci di prodotti leggeri e viene ampiamente utilizzato in campo alimentare e farmaceutico; le sue prestazioni in termini di velocitá e accelerazione sono le migliori nella categoria dell'imballaggio; presenta peró una limitazione di carico massimo pari a 6 kg mentre dispone di massima velocitá lineare raggiungibile pari a 10 m/s e massima accelerazione lineare di 150 m/s.

- **Adept smart controller:**

Figura 5.3: Adept Smart Controller:

Il sistema adept smart controller é un unitá di governo ad alte prestazioni per robot e sistemi di visione; si basa sull' architettura smartservo che consente un risparmio significativo della CPU. L' adept smart controller dispone di un processore veloce e predisposto per essere abbinato a sistemi di visione e porte encoder. Puó comandare fino a 24 assi esterni permettendo applicazioni multi assi e multi robot. L' interfaccia di comunicazione smartservo é basata sulla tecnologia FireWire IEE1394 coerente con il resto del set-up.

5.1.2 Integrazione Progetto di Visione con Sistema DRC

Il sistema di controllo DRC appena introdotto prevede un sistema di visione in grado di calcolare ed elaborare l'angolo di inclinazione di un filo con carico sospeso. Nel caso trattato in tesi, ovvero con liquido in movimento,

la situazione risulta abbastanza analoga in quanto non si controlla piú un angolo su oscillazioni verticali lungo un filo ma si controllano delle oscillazioni orizzontali su un profilo estratto da un liquido in movimento. Alla fine in entrambe le configurazioni la compensazione si basa su un valore di angolo letto, portando quasi in secondo piano la provenienza di questo. Altro motivo di similitudine tra le due configurazioni deriva dal fatto che, come visto in capitolo 4, la dinamica di un liquido in movimento all'interno di un recipiente viene associata a quella di un modello equivalente formato da un pendolo. Tramite pochi accorgimenti infatti é possibile configurare il sistema DRC pure per un controllo sulle oscillazioni di un liquido in movimento.
A questo proposito vengono ora riportati dei test sperimentali con questa configurazione. Il set up rispecchia quello descritto nel precedente paragrafo in cui il contenitore viene fissato all'organo terminale del robot parallelo tramite un supporto a C.
Il test prevede di eseguire una movimentazione a dente di sega sul robot (movimentazione classica in avanti e indietro) e lo studio del modo di oscillare del liquido in presenza o meno del controllo DRC attivato. Sono stati eseguiti infatti due test, il primo con DRC disattivato mentre il secondo con il controllo attivo.
La figura seguente riporta l'andamento dell'angolo misurato nel tempo dal sistema di visione nel caso di prima configurazione ovvero senza il controllo DRC.

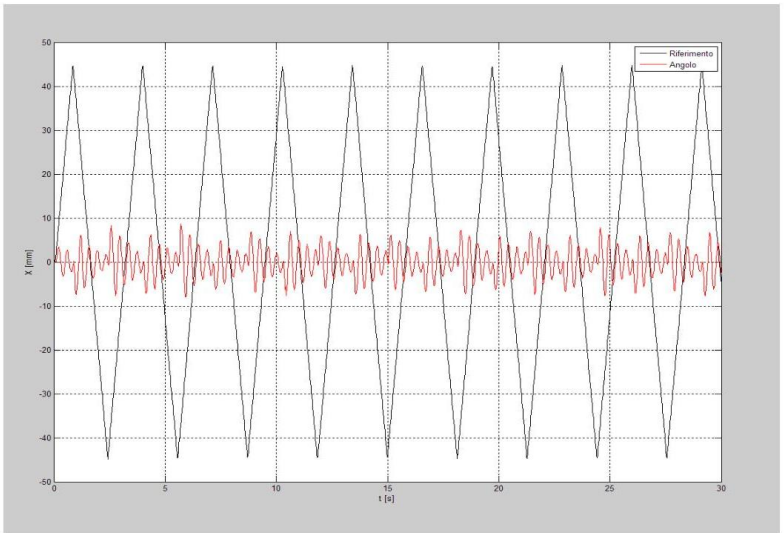


Figura 5.4: Test con legge a dente di sega senza DRC

Com'é facile vedere le oscillazioni risultano lineari e ben definite ad ogni cambio di direzione a fine percorso. Sono presenti dei leggeri smorzamenti naturali che riducono le oscillazioni leggermente tra un cambio di direzione e l'altro.

La figura seguente riporta invece lo stesso test in presenza di controllo DRC:

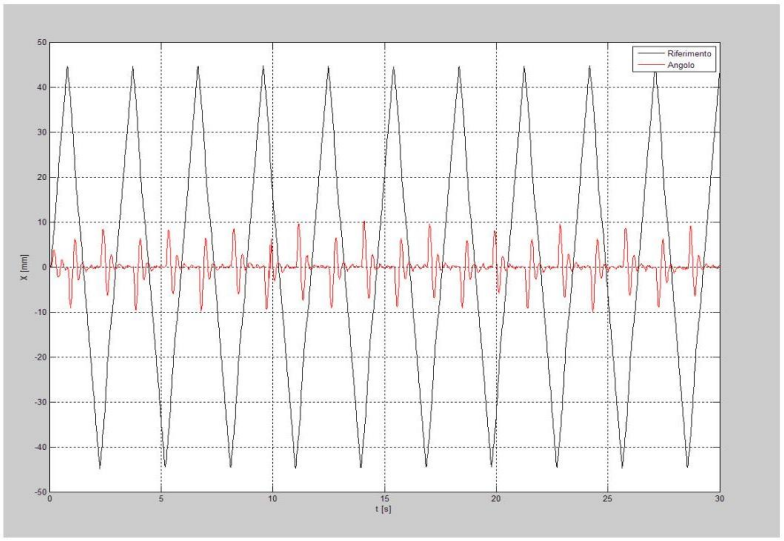


Figura 5.5: Test con legge a dente di sega con DRC

Si nota subito il grande contributo dovuto al controllo DRC che correttamente interpreta l'angolo ricevuto dal sistema di visione andando a smorzare le oscillazione dovute ai vari cambi di direzione durante la fase di test. Il liquido risulta infatti molto piú stabile durante la movimentazione quasi come avesse una densitá maggiore.
Per visualizzare meglio il contributo del controllo viene ora riportata un'ulteriore figura in cui vengono sovrapposti i due andamenti di angolo:

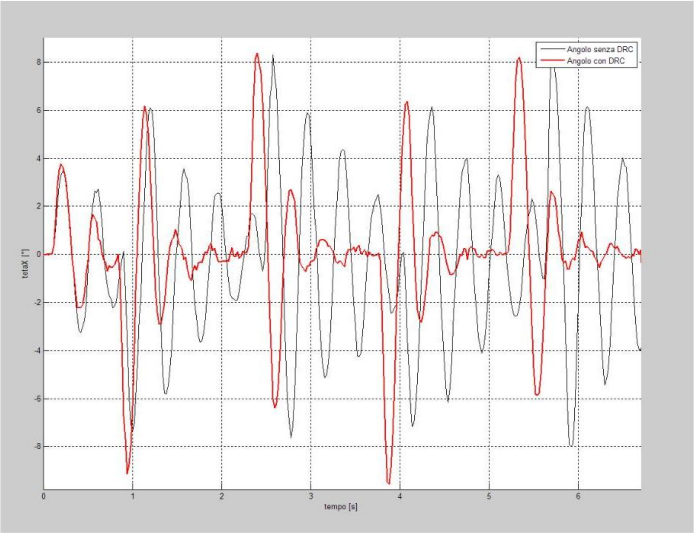


Figura 5.6: Sovrapposizione dei due andamenti

In quest'ultima rappresentazione viene visualizzato molto bene lo smorzamento delle oscillazioni dovuto al controllo DRC in cui il liquido tende a raggiungere la condizione di quiete molto rapidamente.
Il sistema di visione in aggiunta al controllo DRC funziona adeguatamente portando giá dai primi test dei buoni risultati di smorzamento delle oscillazioni anche con diverse leggi di moto applicate oltre a quella a dente di sega.

Al fine di rendere ancora piú chiaro il funzionamento del sistema DRC accoppiato con il sistema di visione, vengono riportati ora altri due grafici di andamenti dell'angolo nel caso di recipiente cilindrico (il test precedente veniva eseguito su contenitore quadrato).
La prima figura riporta l'andamento dell'angolo senza il controllo del DRC attivato, mentre nella seconda si ripete la stessa simulazione con il controllo attivo.

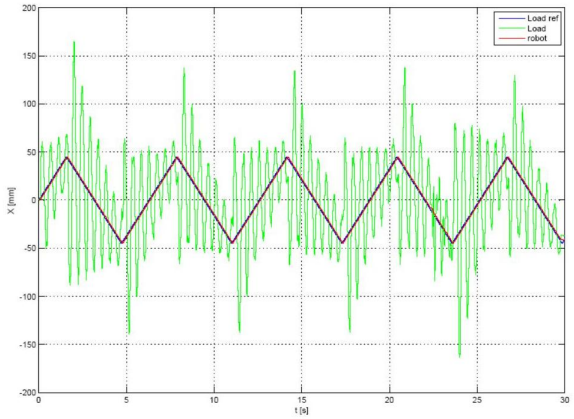


Figura 5.7: Contenitore Cilindrico con legge a dente di sega

La seconda con controllo attivo:

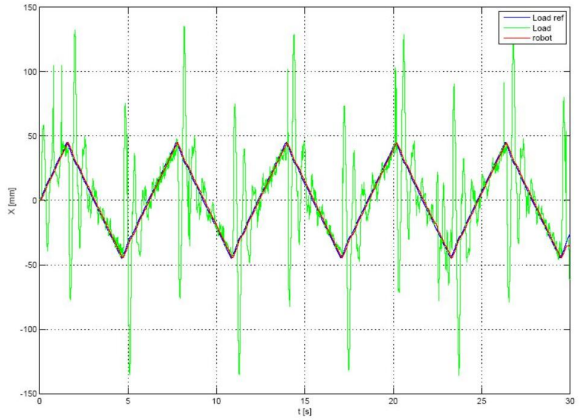


Figura 5.8: Contenitore Cilindrico con legge a dente di sega con DRC

Anche in questa configurazione é possibile notare il grande contributo del controllo del DRC per lo smorzamento delle oscillazione dopo ogni cambio di direzione.

5.1.3 Studio dei modi di vibrare

L'oscillazione di liquidi é un fenomeno non lineare. Il profilo della superficie del liquido in un contenitore puó essere considerato come sovrapposizione di piú modi di oscillare, che dipendono dalla profonditá del liquido, dalla geometria del recipiente e dalle caratteristiche dell'eccitazione esterna. Il modello a parametri distribuiti é ben noto come un valido strumento per l'analisi e la simulazione del comportamento dinamico dei fluidi. Tuttavia, essendo espresso per mezzo di equazioni molto complicate, non é adatto come modello per il progetto del sistema di controllo. Si preferisce quindi utilizzare un modello piú semplice, come quello a parametri concentrati.
Muovendo lateralmente un contenitore riempito parzialmente di liquido, si forma sulla superficie dello stesso un'onda oscillante:

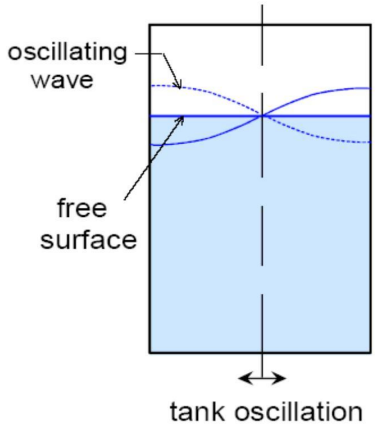


Figura 5.9: Esempio di modello

Il movimento ondulatorio della superficie del liquido presenta una frequenza naturale che dipende dal tipo di liquido, dalla forma del recipiente e dall'accelerazione di gravitá. L'effetto dinamico principale di questo fenomeno é un'oscillazione orizzontale del centro di massa del liquido rispetto al contenitore che puó essere rappresentato con un modello meccanico equivalente, in cui un pendolo identifica le oscillazioni del centro di massa del liquido. La frequenza naturale di oscillazione del pendolo varia rispetto all'accelerazione di gravitá tanto quanto varia la frequenza naturale dell'onda oscillante sulla superficie del liquido. Questa é l'onda fondamentale che presenta un picco ed una valle ed ha la frequenza naturale piú bassa tra tutti i modi del sistema

ovvero il modo dominante. Anche altre onde con piú picchi e piú valli con frequenze naturali piú elevate possono presentarsi.
Nell'analisi dei modi di vibrare di un liquido in un recipiente si possono distinguere principalmente due modi vibrare: un modo simmetrico e un modo asimmetrico. Al fine di rendere piú chiara la spiegazione si riportano ora due immagini dei possibili modi elencati:

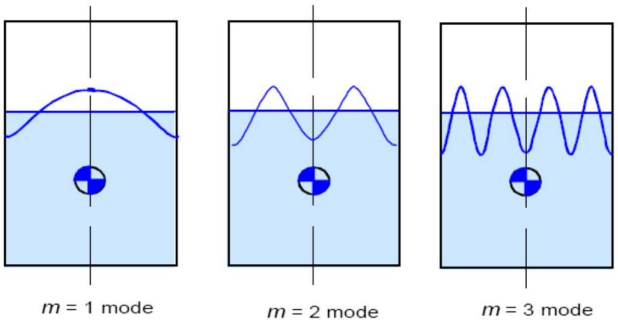


Figura 5.10: Modi Simmetrici

I primi modi riportati sono di tipo simmetrico e la caratteristica risulta ben chiara visto la simmetria della figura dell'onda rispetto al centro del recipiente. La prossima figura invece riporta i modi asimmetrici:

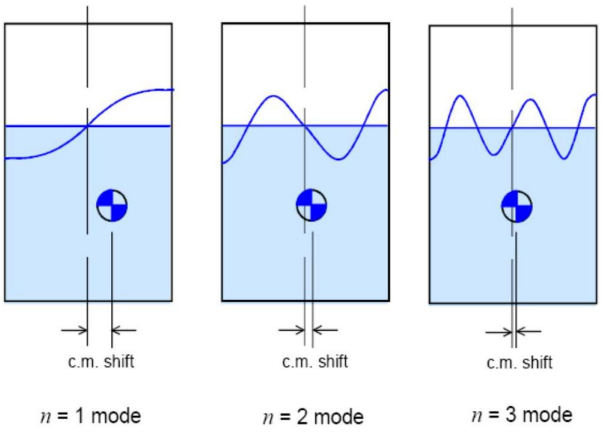


Figura 5.11: Modi Asimmetrici

Nella Figura 5.11 vengono riportati i modi asimmetrici nel caso di liquido dentro contenitore.
Vengono inserite ora due ulteriori figure reali dei due tipi di modi di vibrare acquisite durante le fasi di test in laboratorio:

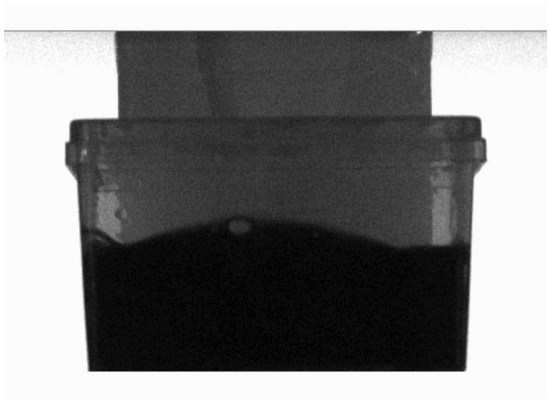


Figura 5.12: Modo simmetrico su contenitore quadrato

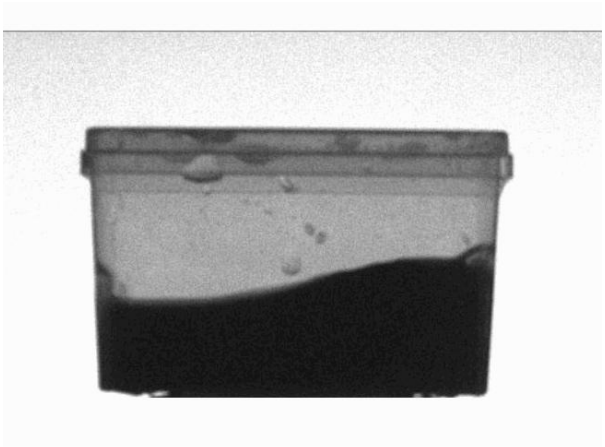


Figura 5.13: Modo assimmetrico su contenitore quadrato

Per quanto riguarda la difficoltá di rilevamento del modo di vibrare questa risulta piú elevata nella versione simmetrica. In questa versione l'onda creata, essendo simmetrica rispetto al centro, sia nella versione di calcolo tramite

momenti che con retta interpolante non risulta rilevabile con angolo di inclinazione trovato di valore nullo. Nel caso dei momenti e quindi tramite ellisse la figura geometrica sará inscritta in un ellisse con angolazione quasi nulla, come se il liquido fosse in quiete. Anche nel calcolo tramite retta interpolante si verifica lo stesso problema. Questo secondo metodo, basandosi sui punti estremi della figura ottenuta dopo il tracciamento del profilo, andrá a creare una retta quasi orizzontale non riuscendo a interpretare esattamente l'inclinazione giusta. Per quanto riguardano i modi asimmetrici invece questi non portano problemi di identificazione vista la natura geometrica del loro movimento. Con entrambi i metodi é ben possibile rilevare l'angolo di inclinazione. Al fine di spiegare in modo esaustivo il problema vengono ora riportati due esempi della problematica:

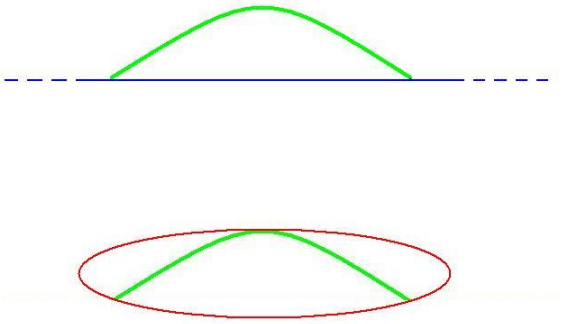


Figura 5.14: Esempio di errata lettura su retta interpolante e ellisse

Le due figure riportate portano esempio del primo modo di vibrare (m=1) dei modi simmetrici, in questa situazione, nemmeno uno dei due metodi di calcolo é in grado di rilevare il movimento del liquido. La stessa situazione si verifica per i modi simmetrici di ordine superiore.

5.1.4 Limiti metodo retta interpolante

Visto il problema dei modi simmetrici viene ora proposto piú nel dettaglio il metodo di calcolo dell'inclinazione del liquido in vista frontale tramite retta interpolante. Come giá ribadito tale procedura si basa solamente sulla posizione dei punti piú estremi, a destra e a sinistra, del profilo del liquido.

Tramite questi due punti poi sará identificata una retta passante attraverso la quale ne verrá poi identificato il coefficiente angolare e la relativa pendenza. Questo metodo, molto semplice e veloce nell'esecuzione, presenta peró diversi difetti e limiti. Per prima cosa, come giá introdotto in paragrafo, la procedura non é in grado di identificare i modi simmetrici del liquido vista la loro configurazione geometrica, a tal fine vengono riportate in seguito le tue tipologie di modi di vibrare (le stesse figure di paragrafo precedente) con l'aggiunta della retta interpolante nei vari casi.

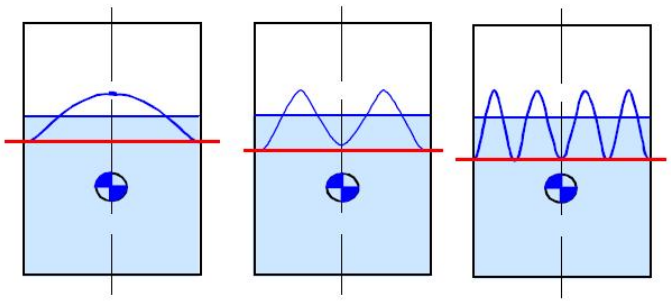


Figura 5.15: Retta interpolante su modi simmetrici

La prima figura riporta il limite del programma visto dal metodo nei modi simmetrici mentre la seconda evidenzia il funzionamento corretto con i modi asimmetrici:

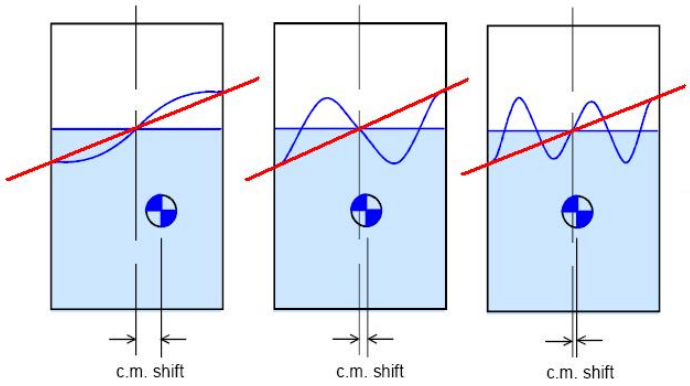


Figura 5.16: Retta interpolante su modi asimmetrici

Una possibile soluzione al problema incontrato con i modi simmetrici potrebbe consistere nel dividere il profilo finale del liquido in piú parti, ed evidenziare quindi piú tratti di questo e su ognuno calcolarne l'angolo di inclinazione tramite retta interpolante. Poi tramite un analisi dei valori ottenuti si potrebbe determinare anche la presenza di modi simmetrici o comunque di oscillazioni che altrimenti non sarebbero rilevate vista la sola analisi dei punti esterni. Questo tipo di procedura peró, com'é facile intuire, introduce una grande quantitá di calcolo e quindi un calo di prestazioni in termini di velocitá. L'elaborare una quantitá quattro volte (per esempio considerando di suddividere il tratto di profilo in quattro parti) piú grande di dati comporta un onerositá computazionale non indifferente e viste le necessitá di prestazione del programma la soluzione, sebbene di migliore analisi sui modi, non rispecchia tutte le specifiche di consegna.
Confrontando il metodo tramite interpolazione con quello dei momenti in alcuni casi puó rivelarsi migliore rispetto a quello con ellisse. Con certe configurazioni particolari di movimento di liquido si possono creare delle figure geometriche che solamente tramite interpolazione possono essere rilevabili. Un esempio di frame con queste caratteristiche viene riportato in figura sotto:

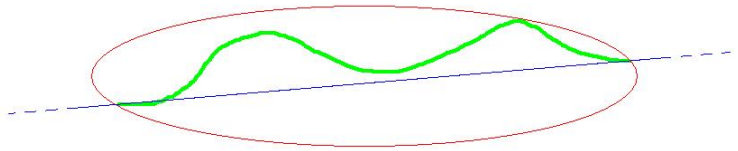


Figura 5.17: Confronto tra i due metodi

Come si puó vedere dalla figura, solamente il calcolo dell'angolo con interpolazione risulta valido in quando l'ellisse, ottenuto per inscrivere all'interno la figura estratta del profilo, presenta angolo di inclinazione nullo.
Il metodo tramite interpolazione implementato presenta peró complessivamente molti piú problemi di stabilitá rispetto a quello con i momenti. Un grande limite deriva dal fatto che la retta viene costruita sui punti esterni del profilo. In quella zona, ovvero agli estremi del recipiente, spesso il liquido presenta problemi di visibilitá che saranno visti nel successivo paragrafo del capitolo. In quelle due zone infatti, sia nel caso di contenitore cilindrico che rettangolare e/o quadrato, la luce viene riflessa diversamente rispetto alla

superficie piana nel corpo del recipiente causando continue variazioni di lettura. Anche in condizioni di quiete e quindi con liquido fermo, sono presenti delle piccole oscillazioni nella lettura dell'angolo che possono compromettere un possibile controllo del liquido. Per questo motivo viene suggerito l'utilizzo della versione con codice di calcolo basato sui momenti in quanto piú stabile e priva di problemi di lettura.

5.1.5 Configurazione a piú telecamere

Un possibile e interessante sviluppo futuro del sistema potrebbe essere l'integrazione del sistema con un ulteriore telecamera di visione. Fino ad adesso il sistema prevede la visione solamente su un lato del contenitore, in particolare su quello relativo alla movimentazione rispetto all'asse x del manipolatore. L'aggiunta di una telecamera consente la misurazione del movimento del liquido pure rispetto all'asse y del manipolatore. In questa configurazione si avrebbe una comprensione migliore del tutto e la possibilitá di studiare pure le movimentazioni e i modi sull'altro lato, anche al variare del recipiente utilizzato. Per quanto riguarda la configurazione software il problema non si pone in quanto il codice, realizzato con l'utilizzo delle threads, é stato progettato per un eventuale inserimento di una o piú videocamere. Nell'eventualitá di questa aggiunta il codice automaticamente, rilevando il numero totale delle telecamere collegate, creerebbe un numero di threads pari a quello dei dispositivi collegati le quali poi funzionerebbero in modo distinto tra loro. L'eventuale integrazione con piú videocamere non sarebbe quindi un problema, come non lo sarebbe nemmeno per quanto riguarda l'estrazione del profilo del liquido. Per com'é costruito il codice ogni thread lavorerebbe in parallelo e ognuna sarebbe in grado di estrarre autonomamente il profilo dai frame ricevuti. Vista la simmetria del contenitore il problema dell'estrazione del profilo non si pone visto che, sia il profilo cilindrico che quadrato visto in modo ortogonale, rimane lo stesso dal punto di vista geometrico. Un possibile problema potrebbe essere dovuto al profilo di supporto che sorregge il recipiente. Vista la sua forma, nella vista laterale, si avrebbe sullo sfondo delle immagini acquisite pure il supporto stesso. Tramite una adeguata sogliatura binaria e posizionamento corretto di luci in aggiunta alla colorazione bianca del supporto a C, non si presentano problemi di disturbo nell'analisi dei fotogrammi ricevuti. A questo proposito sono stati fatti dei test di prima verifica con questa configurazione. Il sistema risponde perfettamente alla modifica inserita.

A tal proposito viene riportata sotto una figura di esempio con entrambe le elaborazioni delle immagini acquisite.

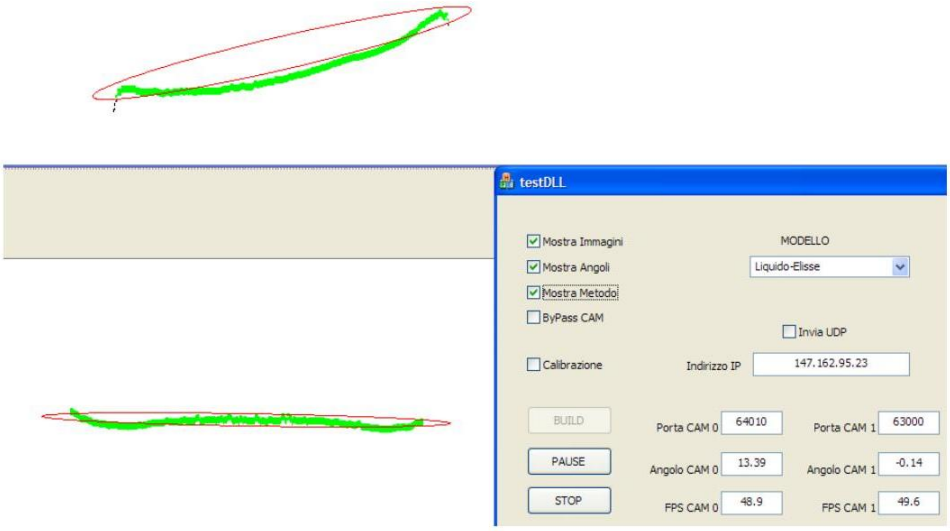


Figura 5.18: Esempio funzionamento a 2 telecamere

Come si puó ben vedere in figura (dove sono riportate entrambe le immagini durante l'esecuzione di un test di movimento) entrambe le visioni riescono ad estrarre il profilo del liquido senza problemi e errori. Anche la lettura dell'angolo non riporta errori e la velocitá di calcolo rimane inalterata rispetto alla versione a singola telecamera, costante con media sui 50Hz. Questo tipo di configurazione permette quindi un movimento su entrambi gli assi, sia x che y, anche contemporaneamente. Analizzando il movimento solamente rispetto all'asse x, ovvero solamente rispetto alla telecamera iniziale, con l'aggiunta della vista laterale, con questo tipo di movimentazione l'angolo di lettura sulla parte laterale dovrebbe essere nullo o quasi anche in caso di arresto o di cambio di direzione.

La figura di seguito porta un esempio reale della situazione appena descritta, con oscillazione lungo asse x e cambio di livello su asse y:

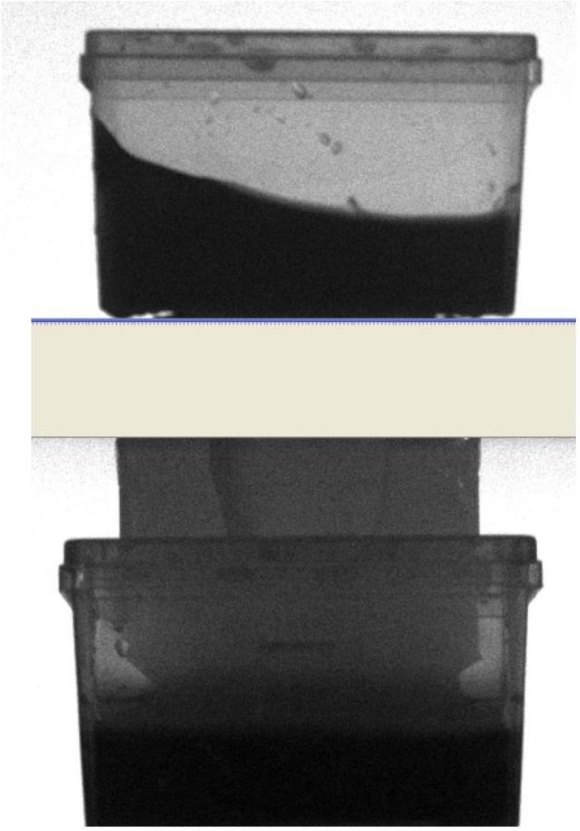


Figura 5.19: Esempio visione su 2 assi

In questa situazione infatti, nel caso di contenitore quadrato, il liquido dovrebbe solamente salire e scendere di livello mentre le oscillazioni dovrebbero essere visibili solamente dal lato principale. E' proprio questo che si verifica in Figura 5.19 visto anche il valore di lettura degli angoli del fotogramma, segno di ulteriore bontá sulla realizzazione del sistema.

5.2 Conclusioni

5.2.1 Problematiche di Visione

Il primo paragrafo si concentra su uno dei problemi piú rilevanti del sistema ovvero i problemi classici relativi alla visione. La visione in generale viene considerata come un ottima soluzione non invasiva per ottenere delle misurazioni in diversi ambienti anche poco accessibili. L'utilizzo di un sensore, anche nel caso trattato nella tesi, tralasciando i costi di implementazione, risulta un approccio troppo invasivo e di difficile realizzazione. Una configurazione con sensoristica ha dalla sua parte peró una maggiore prestazione e velocitá visto il trattare di semplici segnali (che possono essere binari o meno) variabili nel tempo. Questa variazione sará poi interpretata adeguatamente e convertita in informazioni utili. Questa grande performance che si ottiene con utilizzo di sensori peró decade in ambiti poco accessibili e di difficile lavoro. In questi casi, a scapito della velocitá, viene utilizzato un sistema di visione. Un sistema di questo tipo, oltre ai limiti di velocitá delle telecamere, comporta peró la progettazione di sistema di calcolo e interpretazione delle immagini acquisite in tempo reale. Si dovrá quindi implementare un codice di calcolo per il caso considerato e studiare dei metodi di interpretazione dei fotogrammi. Questa procedura, oltre al lavoro dell'utente, ne risente sulle prestazioni del sistema finale, costretto a elaborare continuamente e in tempo reale ogni singolo fotogramma ricevuto.
Un esempio lampante sui vantaggi della visione e delle sue potenzialitá, oltre al caso studiato nel leggere e trovare l'angolo di oscillazione e quindi il movimento di un fluido in un recipiente, si puó riscontrare sulla rivelazione degli ostacoli. Questo ambito, ben noto nella robotica, rispetto alla configurazione con sensori di forza o tramite celle di carico si interpreta (sempre con i problemi di minore velocitá fin qui spiegati) molto meglio con un sistema di visione. Con questa configurazione infatti é possibile prevedere con un certo anticipo, sondando lo spazio di lavoro, la presenza di un ostacolo indesiderato nel percorso andandolo a evitare con precisione. Nel caso di configurazione con sensore invece il sistema dovrá scontrarsi con l'oggetto e procedere poi tramite i segnali ricevuti a trovare un percorso alternativo di movimento. Nel confronto tra i due, basta pensare alla grande quantitá di calcolo del sistema di visione per l'elaborazione delle immagini in tempo reale e la definizione della posizione e dimensione dei vari ostacoli per capirne la minore prestazione in velocitá.
L'altro grande problema della visione riguarda la luce e intensitá luminosa dell'ambiente di lavoro. L'intensitá luminosa variabile infatti disturba in modo rilevante un sistema di visione in quanto tale solitamente viene progettato

per condizioni di lavoro quasi statiche (in termini di luce) e una variazione di questa puó portare ad una diversa interpretazione dell'oggetto. Il codice di calcolo, non essendo intelligente, interpreta tutti i dati basandosi su dei valori di intensitá luminosa dei pixel e situazioni di riflesso o luce diversa possono portare a risultati differenti, anche con una stessa configurazione di lavoro. Anche nel caso trattato in tesi il problema é stato riscontrato in quanto anche solo per diverse condizioni metereologiche il risultato delle analisi esce diverso. Una luce da diversa fonte puó infatti produrre immagini differenti soprattutto dovute dalla posizione della fonte luminosa. Anche il fatto di utilizzare contenitori di materiali diversi condiziona questa situazione. La presenza di contenitori opachi o lucidi varia il modo di riflettere la luce portando piú o meno disturbo in visione. Anche la forma dello stesso porta a situazioni diverse; un recipiente sferico o cilindrico riflette la luce in modo differente rispetto ad un recipiente con superficie piana. Solo con la variazione di sogliatura spesso non si ottengono risultati accettabili e non si riesce a distinguere lo sfondo dall'oggetto di interesse.
Per questo motivo, in soluzione al problema posto, si possono inserire nello spazio di lavoro delle lampade con direzione opportuna al fine di evitare indesiderati riflessi. Anche un isolamento tramite barriere aiuta a non far entrare fasci di luce non idonei. Nello specifico della situazione trattata si é ricoperto il profilo di supporto a C del contenitore con del semplice nastro adesivo per ridirezionare la luce imposta dalle lampade in modo opportuno. Sono stati posizionati anche dei monitor sul retro dell'oggetto di interesse, monitor con immagine statica di colore chiaro (giallo solitamente) per evitare, nel range di visione della telecamera, interferenze con tutto ció che é presente dietro allo spazio di lavoro. La telecamera infatti riprende una porzione abbastanza piccola di visuale, in cui tutto lo sfondo visibile corrisponde alla dimensione del monitor, sulla quale sará posizionato l'organo terminale e il recipiente nel caso studiato.
Tramite questi accorgimenti e con un adeguata sogliatura dell'immagine si possono ottenere risultati buoni con ogni tipo di configurazione di lavoro.

5.2.2 Confronto sulle velocitá

Vista l'analisi delle problematiche dei metodi realizzati e una trattazione sulla velocitá di questi viene ora introdotto un paragrafo specifico sull'analisi delle prestazioni delle versioni prodotte del software. A livello commerciale programmi di questo tipo o comunque di simile utilizzo, si possono reperire ma presentano quasi sempre difetti e lacune nel campo della velocitá di calcolo. Per questo motivo si é cercato di realizzare un sistema di calcolo molto veloce e in grado di analizzare in modo fluido l'andamento delle variazioni

degli oggetti di studio (nel caso trattato in tesi riguardanti liquido in movimento). Un analisi del movimento del liquido troppo lenta, e quindi a tratti molto distanti tra loro con un intervallo molto ampio tra un frame e l'altro, non riuscirebbe a determinare un andamento continuo dei valori di angolo comportando una grande perdita di informazione. I programmi commerciali di questo tipo lavorano solitamente sui 10Hz ovvero sono in grado di produrre 10 risultati utili in un secondo. Questa velocitá non é quasi mai sufficiente per un analisi dettagliata in casi di studio come quello trattato in tesi e si necessita a tal proposito una maggiore velocitá di esecuzione.
Il software realizzato, prima di raggiungere la forma finale e completa, ha subito diverse evoluzioni e quindi anche diverse prestazioni. Una prima versione finita e funzionante presentava una velocitá pari a 25Hz, doppia rispetto a quella dei normali software in commercio, ma ancora inferiore a quella dell'obiettivo di progetto. Questa prima versione utilizzava dei metodi standard della libreria OpenCV e cvBlobs, metodi peró molto standard e pertanto poco performanti. La presenza di metodi giá implementati e realizzati semplifica molto spesso il lavoro e il tempo di realizzazione dei codici di calcolo ma produce anche una velocitá di esecuzione non cosi ottimale.
A tal merito viene ora riportato il codice della versione iniziale che utilizza il metodo dei momenti con l'estrazione del profilo realizzata con i metodi forniti dalle librerie standard al fine di valutarne i problemi e di introdurre la soluzione successiva per migliorarne le prestazioni:

```
// Estrazione profilo con metodo di prima soluzione

CvScalar color;
CvScalar bianco;
bianco.val[0]=255;

for(int i=0;i<(int)width;i++)
{
      for(int j=0;j<(int)height;j++)
      {
            color=cvGet2D(p_Data->Image_bn, j,i);
            if((unsigned char)(color.val[0])==0)
            {
                  for(int k=j+(p_Data->thickness);k<(int)height;k++)
                  {
                        cvSet2D(p_Data->Image_bn,k,i,bianco);
                  }
                  j=height;

            }
      }
}
```

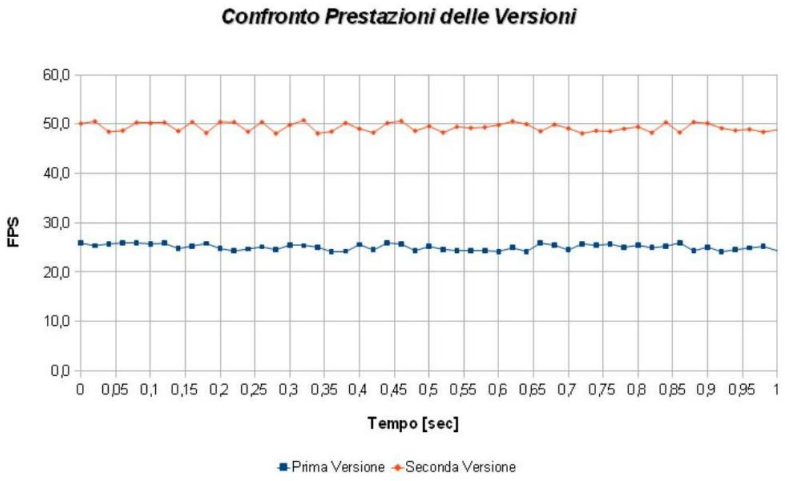
Figura 5.20: Prima versione con metodi standard

Come si puó ben vedere il codice utilizza giá metodi implementati e appare molto semplice nella sua costruzione. Questa semplice realizzazione peró riporta tutti i limiti di velocitá visti sopra. La soluzione adottata per il problema é stata ottenuta con l'utilizzo dell'aritmetica dei puntatori. Nei linguaggi di programmazione, l'espressione aritmetica dei puntatori si riferisce a un insieme di operazioni aritmetiche applicabili sui valori di tipo puntatore. Tali operazioni hanno lo scopo di consentire un alto livello di flessibilitá nell'accesso a dati omogenei conservati in posizioni contigue della memoria (per esempio array o immagini come nel caso della visione). Sfruttando questa caratteristica di organizzazione in memoria delle immagini spiegata nel primo capitolo é possibile accedere in modo diretto al singolo valore di pixel di queste ottenendo prestazioni di gran lunga maggiori rispetto ad un utilizzo normale. Con questa modifica il codice risulta in grado di lavorare a 50Hz ovvero di produrre 50 risultati utili in un secondo. Questa soluzione soddisfa pienamente le esigenze di progetto e l'alta velocitá raggiunta é sufficiente ad ottenere andamenti fluidi delle variazioni dell'angolo. Viene ora proposto, come per il caso precedente, il codice finale operante per l'estrazione del profilo tramite puntatori:

```
// Codice per estrazione profilo liquido
unsigned char value;

for(int i=0;i<(int)width;i++){
	for(int j=0;j<(int)height;j++){

		value = *((unsigned char *)(p_Data->Image_bn->imageData + p_Data->Image_bn-
		>widthStep*j)+i);

		if(value==0){
			for(int k=j+(p_Data->thickness);k<(int)height;k++){
				*((unsigned char *)(p_Data->Image_bn->imageData + p_Data-
				>Image_bn->widthStep*k)+i)=255;
			}
			j=height;
		}
	}
}
```

Figura 5.21: Versione finale con puntatori

L'utilizzo dei puntatori per quanto possa incrementare la velocitá deve essere ben accurato. Questa procedura puó dare esiti rischiosi se non ben curata. La possibilitá di scrivere e leggere direttamente in memoria é operazione delicata e rischiosa, una differente scrittura o su cella sbagliata puó compromettere (e anche con difficile fase di debug) l'esito del programma. Tale soluzione presenta vari pro e contro, ma se ben implementata permette di ottenere grandi velocitá di esecuzione. Il grafico riportato sotto mostra alcuni valori di frame rate acquisiti durante l'esecuzione di entrambe le versioni di software:



Il grafico riporta l'andamento della velocitá in Hertz delle due versione, quella iniziale e quella finale. La differenza di prestazione tra le due versioni é ben visibile: la velocitá risulta praticamente raddoppia con l'introduzione dell'aritmetica dei puntatori. Nell'ultima versione viene inoltre inclusa la parte di codice di filtro per sporcare l'immagine introdotta nel capitolo 3. Questa aggiunta porta ad un incremento di prestazioni di circa 1Hz.

Elenco delle figure

Appendice A

Codice di Calcolo

```
// visDLL.cpp: definisce le funzioni esportate per l'applicazione DLL.
#include "stdafx.h"
#include "visDLL.h"

// Variabili esportate
VISDLL_API int NumCam = 0;
VISDLL_API int NumVer = -1;
VISDLL_API bool sending = FALSE;
VISDLL_API bool running = FALSE; //TRUE;
VISDLL_API bool calibration = FALSE;
VISDLL_API bool MostraMetodo = FALSE;
VISDLL_API HANDLE* h_Thread = NULL;
VISDLL_API Data* p_ThreadData = NULL;

// Funzioni esportata.
#ifdef __cplusplus
extern "C" {
#endif


/* Definizione della thread*/
DWORD WINAPI thread(LPVOID lpParameter)
{
//converto parametro
Data* p_Data = (Data *) lpParameter;

//per sfasare i threads ed evitare sovrapposizioni
Sleep( 1000 *(p_Data->cam_index) + 1 );
```

```
/*Creazione file di testo per salvataggio valori acquisiti*/
char nome_file[BUFFER_SIZE] = "";
FILE* file;
errno_t err;

time_t now ;
now = time( 0 );
struct tm* date;
date = localtime( &now );

sprintf( nome_file, "LOG_%d_%d_%d_%d_%d_CAM%d_LIQUIDO.txt",
date->tm_mday, date->tm_mon+1,
date->tm_year+1900, date->tm_hour, date->tm_min, p_Data->cam_index );

err = fopen_s( &file, nome_file, "a+");
if (err == 0)
fprintf( file, "CENTRO X\tCENTRO Y\tACQUISITO\tCOMPENSATO\tSALVATO\n");
else
MessageBoxA( NULL, (LPCSTR) "Opening file .txt error!",
NULL, MB_OK | MB_ICONWARNING );

int VERSIONE=NumVer;
/*
0 = PENDOLO CON FILO
1 = LIQUIDO CON ELISSE
2 = LIQUIDO CON RETTA INTERPOLANTE
*/

int cicli = 0;
long long int tick_frame;

//SOCKET UDP
WSADATA wsa;
SOCKET sd;
struct sockaddr_in server;

char error_msg[BUFFER_SIZE] = "";
unsigned char send_buffer[BUFFER_SIZE] = "";
int buffer_length = 0;
```

```
unsigned char* ptr_buffer = NULL;

if (WSAStartup(0x0202, &wsa) != 0)
MessageBoxA( NULL, (LPCSTR) "WSA startup error!",
NULL, MB_OK | MB_ICONWARNING );

sd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (sd == INVALID_SOCKET){
MessageBoxA( NULL, (LPCSTR) "Create socket error!",
NULL, MB_OK | MB_ICONWARNING );
WSACleanup();
}

/* Clear out server struct */
memset((void *)&server, '\0', sizeof(struct sockaddr_in));

/* Set family and port */
server.sin_family = AF_INET;
server.sin_port = htons(p_Data->server_port);
server.sin_addr.S_un.S_addr = inet_addr(p_Data->server_IP);

int server_length = sizeof(struct sockaddr_in);

C1394Camera Camera;

//inizializzo camera
Camera.RefreshCameraList();
Camera.SelectCamera(p_Data->cam_index);
Camera.InitCamera();
Camera.StartImageAcquisition();

/* Tempo di pausa per videocamera seguito da acquisizione dimensioni frame */
cvWaitKey(500);
unsigned long width,height;
Camera.GetVideoFrameDimensions( &width, &height );

/* creo immagini: acquisita, in b/n e con blob */
p_Data->Image = cvCreateImage( cvSize(width,height),
                               IPL_DEPTH_8U, CAM_RES_CHANNELS);
p_Data->Image_bn = cvCreateImage( cvSize(width,height),
                                  IPL_DEPTH_8U, 1);
```

```
p_Data->Image_blobs = cvCreateImage( cvSize(width,height),
                                     IPL_DEPTH_8U, CAM_RES_CHANNELS);

//definizione oggetti delle classi di analisi
CBlobResult blobs;
CBlob* filo = NULL;
CBlob* currentBlob = NULL;

CBlobGetArea CBlobGetArea;
CBlobGetMinX CBlobGetMinX;
CBlobGetMaxX CBlobGetMaxX;
CBlobGetMinXatMinY CBlobGetMinXatMinY;
CBlobGetMinYatMaxX CBlobGetMinYatMaxX;
CBlobGetMaxYatMinX CBlobGetMaxYatMinX;
CBlobGetOrientation CBlobGetOrientation;
CBlobGetXCenter CBlobGetXCenter;
CBlobGetYCenter CBlobGetYCenter;
CBlobGetMajorAxisLength CBlobGetMajorAxisLength;
CBlobGetMinorAxisLength CBlobGetMinorAxisLength;
CBlobGetAxisRatio CBlobGetAxisRatio;

//variabili per gestione dati nel ciclo principale
double offset = 0;
double offset_retta = 0;
double offset_pendolo = 0;
unsigned count_calibration = 0;

double old1 = 0;
double old2 = 0;
double media = 0;

int num_blobs=0;
int colore=255;
int step=1;
double area=0;

double angolo=0;
double angolo_temp=0;
double angolo_retta=0;
double angolo_pendolo=0;
```

```
//ciclo principale
while(running)
{
tick_frame = cvGetTickCount();

Camera.AcquireImage();
Camera.getRGB((unsigned char*)(p_Data->Image->imageData),
(p_Data->Image->height*p_Data->Image->width*3));

cvSplit( p_Data->Image, p_Data->Image_bn, NULL, NULL, NULL);

cvThreshold( p_Data->Image_bn, p_Data->Image_bn,
p_Data->livello_bn, 255, CV_THRESH_BINARY );

/*SE IL MODELLO E' QUELLO CON IL LIQUIDO CALCOLO PROFILO E FACCIO PULIZIA*/
if(VERSIONE == 1 || VERSIONE == 2){
// filtro per sporcare immagine
cvErode(p_Data->Image_bn,p_Data->Image_bn,NULL,1);

//analizzo immagine per blobs
blobs = CBlobResult(p_Data->Image_bn, NULL, 255);
num_blobs = blobs.GetNumBlobs();

//Filtro pulizia blob piccoli
double puntox,puntoy;
for(int i=0;i<num_blobs;i++){
filo = blobs.GetBlob(i);
area=CBlobGetArea(*filo);
if(area<300)
filo->FillBlob(p_Data->Image_bn,CV_RGB(0,0,255));
if(area==0){
puntox=CBlobGetXCenter(*filo);
puntoy=CBlobGetYCenter(*filo);
(p_Data->Image_bn->imageData+int(puntoy)*step)[int(puntox)]=255;}
}

/*Estrazione profilo del liquido*/
int pixel=0;
unsigned char value;

for(int i=0;i<(int)width;i++){
```

```
for(int j=0;j<(int)height;j++){
value = *((unsigned char *)
(p_Data->Image_bn->imageData + p_Data->Image_bn->widthStep*j)+i);

if(value==0){
for(int k=j+(p_Data->thickness);k<(int)height;k++){
*((unsigned char *)
(p_Data->Image_bn->imageData + p_Data->Image_bn->widthStep*k)+i)=255;
}
j=height;
}
}
}


blobs = CBlobResult(p_Data->Image_bn, NULL, 255);
num_blobs = blobs.GetNumBlobs();
p_Data->num_blobs = blobs.GetNumBlobs();
}// Fine codice di pulizia e calcolo profilo in modello con liquido

//filtro i blob per area
blobs = CBlobResult(p_Data->Image_bn, NULL, 255);
blobs.Filter( blobs, B_EXCLUDE, CBlobGetArea, B_LESS, p_Data->area_l );
num_blobs = blobs.GetNumBlobs();
p_Data->num_blobs = blobs.GetNumBlobs();

/*CODICE ANGOLO TRAMITE RETTA INTERPOLANTE (CASO VERSIONE = 1)*/
CvPoint2D64f punto1,punto2;
if(VERSIONE==2){
double ipotenusa;
if(num_blobs==1){
filo = blobs.GetBlob(0);
punto1.x=CBlobGetMinX(*filo);
punto2.x=CBlobGetMaxX(*filo);
punto1.y=CBlobGetMaxYatMinX(*filo);
punto2.y=CBlobGetMinYatMaxX(*filo);
ipotenusa=sqrt((punto2.x-punto1.x)*(punto2.x-punto1.x)+
(punto1.y-punto2.y)*(punto1.y-punto2.y));
angolo_retta=asin(fabs((punto1.y-punto2.y))/ipotenusa);
angolo_retta=angolo_retta*180/3.1415;
}
```

```
}

/*RITORNO IMMAGINE DA B/N A RGB*/
cvMerge( p_Data->Image_bn, p_Data->Image_bn,
p_Data->Image_bn, NULL, p_Data->Image_blobs );

/*NEL CASO DI MODELLO CON LIQUIDO COLORO IL PROFILO DI VERDE*/
if(VERSIONE==1 || VERSIONE==2)
if (blobs.GetNumBlobs()){
filo = blobs.GetBlob(0);
filo->FillBlob(p_Data->Image_blobs, CV_RGB(0,255,0));
}

/*CALCOLO ANGOLO LIQUIDO E DISEGNO ELISSE (VERSIONE = 1)*/
if(VERSIONE==1){
num_blobs = blobs.GetNumBlobs();
double or=0;
if(num_blobs != 1)
angolo=angolo_temp;
else{
filo=blobs.GetBlob(0);
or=CBlobGetOrientation(*filo);
angolo=or-180;
angolo_temp=angolo;
}


if(blobs.GetNumBlobs() && MostraMetodo==TRUE){
CvPoint centro;
centro.x=(int)CBlobGetXCenter(*filo);
centro.y=(int)CBlobGetYCenter(*filo);
CvSize assi;
assi.height=(int)CBlobGetMinorAxisLength(*filo);
assi.width=(int)CBlobGetMajorAxisLength(*filo);
cvEllipse(p_Data->Image_blobs,
centro,assi,angolo,0,360,CV_RGB(255,0,0),1,8,0);
}
}

/*DISEGNO RETTA INTERPOLANTE (VERSIONE = 2)*/
if(VERSIONE==2 && MostraMetodo==TRUE)
```

```
if(blobs.GetNumBlobs()){
CvPoint pt1,pt2;
pt1.x=(int)punto1.x;
pt2.x=(int)punto2.x;
pt1.y=(int)punto1.y;
pt2.y=(int)punto2.y;
cvLine(p_Data->Image_blobs,pt1,pt2,CV_RGB(0,0,255),1,8,0);
}

/*CALCOLO ANGOLO IN CASO DI MODELLO CON PENDOLO*/
if(VERSIONE==0){
p_Data->num_blobs = blobs.GetNumBlobs();

//se diverso da zero passo
if( p_Data->num_blobs > 0 ){
filo = blobs.GetBlob(0);
for ( int i = 0; i < p_Data->num_blobs; i++ ){
currentBlob = blobs.GetBlob(i);
if( CBlobGetAxisRatio(*currentBlob) < CBlobGetAxisRatio(*filo) )
filo = currentBlob;
else
currentBlob->FillBlob( p_Data->Image_blobs, CV_RGB(0,255,0) );
}

if( CBlobGetAxisRatio(*filo) > 0.1 ){
fprintf( file, "\n" );
continue; //passo al prossimo ciclo
}
if(CBlobGetYCenter(*filo) <
((height/2)*.99) || CBlobGetYCenter(*filo) > ((height/2)*1.01) ){
fprintf( file, "\n" );
continue;
}

//evidenzio in ROSSO il blob con rapporto assi X/Y maggiore
filo->FillBlob(p_Data->Image_blobs, CV_RGB(255,0,0) );

//ottengo l'angolo
angolo_pendolo = CBlobGetOrientation(*filo);

if( angolo_pendolo == 0.0 ){
```

```
fprintf( file, "\n" );
continue;
}
if( err==0 ){
fprintf( file, "%.1f\t\t%.1f\t\t%.2f\t\t", CBlobGetXCenter( *filo),
CBlobGetYCenter( *filo), angolo_pendolo);
}

//compensazione discontinuit\'a
if( angolo_pendolo > 180 )
angolo_pendolo = angolo_pendolo - 180;

fprintf( file, "%.2f", angolo_pendolo );
}
} //fine caso pendolo

/*EFFETTUO CALIBRAZIONE DEGLI ANGOLI DI TUTTI E 3 I MODELLI*/
//Azzero conteggio se non c'\'e calibrazione
if (!calibration)
count_calibration=0;

//Calibrazione angolo
if( calibration && (count_calibration < MAX_CALIBRATION) ){
//se inizio calibrazione azzero offset
if (count_calibration==0){
offset=0;
offset_retta=0;
offset_pendolo=0;
}
offset += ( angolo / MAX_CALIBRATION );
offset_retta += ( angolo_retta / MAX_CALIBRATION );
offset_pendolo += ( angolo_pendolo / MAX_CALIBRATION );
count_calibration++;
}

/*IN BASE ALLA CONFIGURAZIONE SCELTA INVIO L'ANGOLO CORRETTO*/
if(VERSIONE==1){
angolo = angolo - offset;
angolo = angolo * GAIN_ANGLE;
p_Data->angolo = (int)angolo;
}
```

```
if(VERSIONE==2){
angolo_retta = angolo_retta - offset_retta;
angolo_retta = angolo_retta * GAIN_ANGLE;
p_Data->angolo = (int)angolo_retta;
}
if(VERSIONE==0){
angolo_pendolo = angolo_pendolo - offset_pendolo;
media = angolo_pendolo;
old2 = old1;
old1 = angolo_pendolo;
angolo_pendolo = angolo_pendolo * GAIN_ANGLE;
p_Data->angolo = (int)angolo_pendolo;
}

/* Spedizione dati ANGOLO via UDP */
if( sending && (sd != INVALID_SOCKET) )
{
//invio INDICE + ANGOLO * GAIN
memset((void *)send_buffer, '\0', BUFFER_SIZE);
ptr_buffer = send_buffer;
buffer_length = 0;

memcpy( (void*) send_buffer, (void*) &cicli, sizeof(cicli));
ptr_buffer = ptr_buffer + sizeof(cicli);
buffer_length += sizeof(cicli);

memcpy( (void*) ptr_buffer, (void*) &p_Data->angolo,
sizeof(p_Data->angolo) );
buffer_length += sizeof(p_Data->angolo);

if( sendto(sd, (const char*)send_buffer, buffer_length,0,
(struct sockaddr *)&server, server_length) == SOCKET_ERROR )
{
sprintf_s(error_msg, BUFFER_SIZE, "Sending error %d", WSAGetLastError() );
MessageBoxA( NULL, (LPCSTR) error_msg, NULL, MB_OK | MB_ICONWARNING );

//In caso di errore chiudo e riapro socket
closesocket(sd);
Sleep(1);
SOCKET sd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
}
```

```
}/* Spedizione terminata */

//salvo tempo analisi frame
tick_frame = cvGetTickCount() - tick_frame;

//p_Data->cicli++;
cicli++;

//calcolo FPS
p_Data->FPS = 1 / (tick_frame/(cvGetTickFrequency()*1000000));

}//fine ciclo principale while

/*CHIUSURA SOCKET*/
closesocket(sd);
WSACleanup();

/*RILASCIO IMMAGINI e BLOBS*/
cvReleaseImage( &p_Data->Image );
cvReleaseImage( &p_Data->Image_bn );
cvReleaseImage( &p_Data->Image_blobs );
blobs.ClearBlobs();

/*CHIUSURA FILE DI TESTO*/
if( file )
fclose(file);

return 0;

}//Fine thread

VISDLL_API void build(void)
{
for(int i =0; i < NumCam; i++){
p_ThreadData[i].cam_index = i;
p_ThreadData[i].area_h = AREA_H;
p_ThreadData[i].area_l = AREA_L;
p_ThreadData[i].livello_bn = LIV_BN;
p_ThreadData[i].TID = 0;
p_ThreadData[i].FPS = 0;
p_ThreadData[i].Image = NULL;
```

```
p_ThreadData[i].Image_bn = NULL;
p_ThreadData[i].Image_blobs = NULL;
p_ThreadData[i].angolo = 0;
p_ThreadData[i].thickness = THICKNESS;
}

//creo thread per gestione camere
for (int j =0; j<NumCam; j++){
h_Thread[j] = (HANDLE) CreateThread(
            NULL,  //  pointer to security attributes
            0,     //  initial thread stack size
(LPTHREAD_START_ROUTINE) thread,   // LPTHREAD_START_ROUTINE
            &p_ThreadData[j],// argument for new thread
CREATE_SUSPENDED, // creation flags
            &p_ThreadData[j].TID// pointer to receive thread ID
);
}

} //fine build

VISDLL_API void checkCam(void)
{
//check iniziale per ottenere numero videocamere
C1394Camera CamTest;
CamTest.CheckLink();
CamTest.RefreshCameraList();
NumCam = CamTest.GetNumberCameras();

//alloco e inizializzo dati per threads
h_Thread = (HANDLE*) calloc( NumCam, sizeof(HANDLE));
p_ThreadData = (Data*) calloc( NumCam, sizeof(Data));
}

#ifdef __cplusplus
}
#endif


// Costruttore di una classe esportata.
// Vedere visDLL.h per la definizione della classe
CvisDLL::CvisDLL()
```

```
{
return;
}
```

Appendice B

Codice di Interfaccia

```
// testDLLDlg.cpp : file di implementazione

#include "stdafx.h"
#include "testDLL.h"
#include "testDLLDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// finestra di dialogo CAboutDlg utilizzata per
visualizzare le informazioni sull'applicazione.

class CAboutDlg : public CDialog
{
public:
CAboutDlg();

// Dati della finestra di dialogo
enum { IDD = IDD_ABOUTBOX };

protected:
virtual void DoDataExchange(CDataExchange* pDX); // supporto DDX/DDV

// Implementazione
protected:
DECLARE_MESSAGE_MAP()
```

```
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD){}

void CAboutDlg::DoDataExchange(CDataExchange* pDX){
CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

// finestra di dialogo CtestDLLDlg

CtestDLLDlg::CtestDLLDlg(CWnd* pParent /*=NULL*/)
: CDialog(CtestDLLDlg::IDD, pParent)
, m_IP(_T("147.162.95.23"))
, m_porta0(_T("64010"))
, m_porta1(_T("63000"))
, m_angolo0(_T(""))
, m_angolo1(_T(""))
, m_FPS0(_T(""))
, m_FPS1(_T(""))
, m_warning0(_T(""))
, m_warning1(_T(""))
{
m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CtestDLLDlg::DoDataExchange(CDataExchange* pDX)
{
//Associo agli oggetti le relative stringhe/variabili
CDialog::DoDataExchange(pDX);
DDX_Control(pDX, IDC_BUILD, m_btnBuild);
DDX_Control(pDX, IDC_START, m_btnStartPauseResume);
DDX_Control(pDX, IDC_STOP, m_btnStop);
DDX_Text(pDX, IDC_IP, m_IP);
DDX_Text(pDX, IDC_PORT0, m_porta0);
DDX_Text(pDX, IDC_PORT1, m_porta1);
DDX_Text(pDX, IDC_ANGOLO0, m_angolo0);
DDX_Text(pDX, IDC_ANGOLO1, m_angolo1);
DDX_Text(pDX, IDC_FPS0, m_FPS0);
```

```
DDX_Text(pDX, IDC_FPS1, m_FPS1);
DDX_Text(pDX, IDC_WARNING0, m_warning0);
DDX_Text(pDX, IDC_WARNING1, m_warning1);
DDX_Control(pDX, IDC_COMBO1, Lista);
}

BEGIN_MESSAGE_MAP(CtestDLLDlg, CDialog)
ON_WM_SYSCOMMAND()
ON_WM_PAINT()
ON_WM_QUERYDRAGICON()
ON_WM_TIMER()
//}}AFX_MSG_MAP
ON_BN_CLICKED(IDC_BUILD, &CtestDLLDlg::OnBnClickedBuild)
ON_BN_CLICKED(IDC_STOP, &CtestDLLDlg::OnBnClickedStop)
ON_BN_CLICKED(IDC_START, &CtestDLLDlg::OnBnClickedStart)
ON_BN_CLICKED(IDC_SHOW_IMG, &CtestDLLDlg::OnBnClickedShowImg)
ON_BN_CLICKED(IDC_CHECK_ANGLE, &CtestDLLDlg::OnBnClickedCheckAngle)
ON_BN_CLICKED(IDC_UDP, &CtestDLLDlg::OnBnClickedUdp)
ON_BN_CLICKED(IDC_CALIBRATION, &CtestDLLDlg::OnBnClickedCalibration)
ON_BN_CLICKED(IDC_BYPASS_CAM, &CtestDLLDlg::OnBnClickedBypassCam)
ON_CBN_SELCHANGE(IDC_COMBO1, &CtestDLLDlg::OnCbnSelchangeCombo1)
ON_BN_CLICKED(IDC_SHOW_METHOD, &CtestDLLDlg::OnBnClickedShowMethod)
END_MESSAGE_MAP()

// gestori di messaggi di CtestDLLDlg

BOOL CtestDLLDlg::OnInitDialog()
{
CDialog::OnInitDialog();

// Aggiungere la voce di menu "Informazioni su..." al menu di sistema.

// IDM_ABOUTBOX deve trovarsi tra i comandi di sistema.
ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
ASSERT(IDM_ABOUTBOX < 0xF000);

CMenu* pSysMenu = GetSystemMenu(FALSE);
if (pSysMenu != NULL){
CString strAboutMenu;
strAboutMenu.LoadString(IDS_ABOUTBOX);
if (!strAboutMenu.IsEmpty()){
```

```
pSysMenu->AppendMenu(MF_SEPARATOR);
pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
}
}

//Impostare l'icona per questa finestra di dialogo.
//Il framework non esegue questa operazione automaticamente
//se la finestra principale dell'applicazione
//non e' una finestra di dialogo.
SetIcon(m_hIcon, TRUE); // Impostare icona grande.
SetIcon(m_hIcon, FALSE); // Impostare icona piccola.

// TODO: aggiungere qui inizializzazione aggiuntiva.

//inizializzo checkbox
CheckDlgButton( IDC_CHECK_ANGLE, BST_UNCHECKED );
CheckDlgButton( IDC_SHOW_IMG, BST_CHECKED );
CheckDlgButton( IDC_UDP, BST_UNCHECKED );
CheckDlgButton( IDC_CALIBRATION , BST_UNCHECKED );

return TRUE;
//restituisce TRUE a meno che non venga impostato
//lo stato attivo su un controllo.
}

void CtestDLLDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
if ((nID & 0xFFF0) == IDM_ABOUTBOX){
CAboutDlg dlgAbout;
dlgAbout.DoModal();
}
else{
CDialog::OnSysCommand(nID, lParam);
}
}

// Se si aggiunge alla finestra di dialogo un pulsante
//di riduzione a icona, per trascinare l'icona sara' necessario
//il codice sottostante. Per le applicazioni MFC che utilizzano
//il modello documento/visualizzazione,
//questa operazione viene eseguita automaticamente dal framework.
```

```
void CtestDLLDlg::OnPaint()
{
if (IsIconic()){
CPaintDC dc(this); // contesto di dispositivo per il disegno

SendMessage(WM_ICONERASEBKGND,
reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

// Centrare l'icona nel rettangolo client.
int cxIcon = GetSystemMetrics(SM_CXICON);
int cyIcon = GetSystemMetrics(SM_CYICON);
CRect rect;
GetClientRect(&rect);
int x = (rect.Width() - cxIcon + 1) / 2;
int y = (rect.Height() - cyIcon + 1) / 2;

// Disegnare l'icona
dc.DrawIcon(x, y, m_hIcon);
}
else
CDialog::OnPaint();
}

// Il sistema chiama questa funzione per ottenere la
// visualizzazione del cursore durante il trascinamento
// della finestra ridotta a icona.
HCURSOR CtestDLLDlg::OnQueryDragIcon()
{
return static_cast<HCURSOR>(m_hIcon);
}

//PULSANTE BUILD
void CtestDLLDlg::OnBnClickedBuild()
{
if(NumVer!=-1){
NumCam = 0;
p_ThreadData = NULL;
h_Thread = NULL;
running = TRUE;
```

```
if( IsDlgButtonChecked( IDC_CALIBRATION ) )
calibration = TRUE;
else
calibration = FALSE;

if( IsDlgButtonChecked( IDC_UDP ) )
sending = TRUE;
else
sending = FALSE;

//ottengo il numero di videocamere collegate
checkCam();
/* Richiama metodo in visDLL.cpp che inizializza le telecamere*/
if(NumCam == 0){
MessageBoxA(NULL,"Nessuna videocamera collegata!",
NULL, MB_OK | MB_ICONERROR);
OnBnClickedStop();
return;
}

//creo nome barre e finestre
sprintf_s( tbarname1, MAXSTRING, "THRESHOLD");
sprintf_s( tbarname2, MAXSTRING, "NO BIGGER");
sprintf_s( tbarname3, MAXSTRING, "NO SMALLER");
sprintf_s( tbarname4, MAXSTRING, "THICKNESS");
for (int i=0; i<NumCam; i++)
sprintf_s(wnd_cams[i], MAXSTRING, "BLOBS CAM %d", i);

//salvo IP e porte
UpdateData( TRUE ); //recupero info da finestra
for( int i=0; i<NumCam; i++)
wcstombs_s(NULL, p_ThreadData[i].server_IP,
BUFFER_SIZE, m_IP.GetString(), BUFFER_SIZE );
p_ThreadData[0].server_port = (unsigned short)_ttoi( m_porta0 );
p_ThreadData[1].server_port = (unsigned short)_ttoi( m_porta1 );

//creo worker threads
/* Richiama metodo in visDLL.cpp che inizializza le thread*/
build();

//inizializzo stato threads
```

```
m_ThreadState = TS_STOPPED;

//se \'e selezionato mostra immagini creo le finestre di visualizzazione
if( IsDlgButtonChecked( IDC_SHOW_IMG ) ){
RECT rc;
HWND hWnd;
//se OK creo finestre e barre
for( int i =0; i<NumCam; i++ ){
cvNamedWindow( wnd_cams[i], CV_WINDOW_AUTOSIZE );
cvCreateTrackbar( tbarname4, wnd_cams[i],
&p_ThreadData[i].thickness, 20, NULL );
cvCreateTrackbar( tbarname3, wnd_cams[i],
&p_ThreadData[i].area_l, 8000, NULL );
cvCreateTrackbar( tbarname2, wnd_cams[i],
&p_ThreadData[i].area_h, 8000, NULL );
cvCreateTrackbar( tbarname1, wnd_cams[i],
&p_ThreadData[i].livello_bn, 255, NULL );

//posiziono finestre
hWnd = (HWND) cvGetWindowHandle( wnd_cams[i] );
//::GetWindowRect( hWnd, &rc );
::GetClientRect( hWnd, &rc );
cvMoveWindow( wnd_cams[i], (int) 5 ,
(int) (5+2.1*i*(rc.bottom-rc.top)) );
}
}

//altrimenti continuo e attivo il pulsante START
m_btnStartPauseResume.SetWindowTextW(_T("START"));

//abilito-disabilito pulsanti
m_btnBuild.EnableWindow( FALSE );
m_btnStartPauseResume.EnableWindow( TRUE );

m_btnStop.EnableWindow( TRUE );
}
else
MessageBoxA(NULL,"Scegliere modello prima di esecuzione",
NULL, MB_OK | MB_ICONERROR);
}
```

```
//PULSANTE STOP
void CtestDLLDlg::OnBnClickedStop()
{
//se sono in pausa, riesumo i threads per consentire
//l'uscita naturale dal ciclo
if( m_ThreadState == TS_PAUSE )
for(int i = 0; i<NumCam; i++)
ResumeThread( h_Thread[i] );

//set condizione per termine threads e attendo termine threads
running = FALSE;
WaitForMultipleObjects( NumCam, h_Thread, TRUE, INFINITE );

//fermo timer
KillTimer( timer );

//pulisco
free(h_Thread);
free(p_ThreadData);
cvDestroyAllWindows();
sprintf_s( tbarname1, MAXSTRING, "");
sprintf_s( tbarname2, MAXSTRING, "");
sprintf_s( tbarname3, MAXSTRING, "");
sprintf_s( tbarname4, MAXSTRING, "");
for (int i=0; i<NumCam; i++)
sprintf_s(wnd_cams[i], MAXSTRING, "", i);

//gestione pulsanti
m_btnStartPauseResume.EnableWindow( FALSE);
m_btnBuild.EnableWindow( TRUE );
m_btnStop.EnableWindow( FALSE );
}

//TOGGLE STATO THREADS E PULSANTE START-PAUSE
//Ritorna un intero corrispondente
//allo stato delle thread dopo aver premuto
INT CtestDLLDlg::ToggleSPRState()
{
/* Se sono in PAUSE ritorno in START*/
if( TS_RESUME == m_ThreadState )
```

```
m_ThreadState = TS_START;
/* Altrimenti passo allo stato dopo di (enum) cioe' PAUSE */
m_ThreadState++;

CString sButtonText = _T("");

switch( m_ThreadState )
{
case TS_START:
sButtonText = _T("PAUSE");
break;
case TS_PAUSE:
sButtonText = _T("RESUME");
break;
case TS_RESUME:
sButtonText = _T("PAUSE");
break;
default:
ASSERT( 0 ); // We shouldn't reach this
}

// Set button text
m_btnStartPauseResume.SetWindowText( sButtonText );

return m_ThreadState;
}

//PULSANTE START
void CtestDLLDlg::OnBnClickedStart()
{
/* abilito pulsante STOP *SOLO* dopo l'avvio */
m_btnStop.EnableWindow( TRUE );
/* controllo lo stato delle thread */
switch( ToggleSPRState() )
{
case TS_START:
for(int i =0; i<NumCam; i++)
ResumeThread(h_Thread[i]);
//faccio partire timer per aggiornare finestra
timer = SetTimer( 1, 100, 0 );
break;
```

```
case TS_PAUSE:
for(int i = 0; i<NumCam; i++)
SuspendThread( h_Thread[i] );
break;

case TS_RESUME:
for(int i =0; i<NumCam; i++)
ResumeThread(h_Thread[i]);
break;

default: //non si dovrebbe arrivare qui
ASSERT( 0 );
}
}

//CHECK MOSTRA IMMAGINI
void CtestDLLDlg::OnBnClickedShowImg()
{
if( IsDlgButtonChecked( IDC_SHOW_IMG ) )
for( int i =0; i<NumCam; i++ ){
cvNamedWindow( wnd_cams[i], CV_WINDOW_AUTOSIZE );
cvCreateTrackbar( tbarname4, wnd_cams[i],
&p_ThreadData[i].thickness, 20, NULL );
cvCreateTrackbar( tbarname3, wnd_cams[i],
&p_ThreadData[i].area_l, 10000, NULL );
cvCreateTrackbar( tbarname2, wnd_cams[i],
&p_ThreadData[i].area_h, 10000, NULL );
cvCreateTrackbar( tbarname1, wnd_cams[i],
&p_ThreadData[i].livello_bn, 255, NULL );
}
else
cvDestroyAllWindows();
}

//CHECK INVIA UDP
void CtestDLLDlg::OnBnClickedUdp()
{
if ( IsDlgButtonChecked( IDC_UDP ) )
sending = TRUE;
else
```

```
sending = FALSE;
}

//CHECK CALIBRAZIONE
void CtestDLLDlg::OnBnClickedCalibration()
{
if( IsDlgButtonChecked( IDC_CALIBRATION ) )
calibration = TRUE;
else
calibration = FALSE;
}

//TIMER
void CtestDLLDlg::OnTimer( UINT timer )
{
/*se il check mostra angoli \'e attivato li aggiorno */
if ( IsDlgButtonChecked( IDC_CHECK_ANGLE ) ){
m_angolo0.Format(_T("%.2f"),
((double) p_ThreadData[0].angolo/GAIN_ANGLE) );
m_angolo1.Format(_T("%.2f"),
((double) p_ThreadData[1].angolo/GAIN_ANGLE) );
}

//aggiorno FPS
m_FPS0.Format(_T("%.1f"), p_ThreadData[0].FPS);
m_FPS1.Format(_T("%.1f"), p_ThreadData[1].FPS);

/* warnings eventuali sul numero dei blobs trovati */
if( p_ThreadData[0].num_blobs > 1 )
m_warning0.Format(_T("TROPPI BLOBS!"));
else{
if ( p_ThreadData[0].num_blobs == 0 )
m_warning0.Format(_T("NESSUN BLOB!"));
else
m_warning0.Format(_T(" "));
}

if( p_ThreadData[1].num_blobs > 1 )
m_warning1.Format(_T("TROPPI BLOBS!"));
else{
if( p_ThreadData[1].num_blobs == 0 )
```

```
m_warning1.Format(_T("NESSUN BLOB!"));
else
m_warning1.Format(_T(" "));
}

//aggiorno info nella finestra
UpdateData(FALSE);

if ( IsDlgButtonChecked( IDC_SHOW_IMG ) )
for(int i = 0; i<NumCam; i++){
if( IsDlgButtonChecked( IDC_BYPASS_CAM ) )
//cvShowImage( wnd_cams[i], p_ThreadData[i].Image_bn );
cvShowImage( wnd_cams[i], p_ThreadData[i].Image );
else
cvShowImage( wnd_cams[i], p_ThreadData[i].Image_blobs );
}

if( IsDlgButtonChecked( IDC_SHOW_METHOD ) )
MostraMetodo = TRUE;
else
MostraMetodo = FALSE;
}

void CtestDLLDlg::OnCbnSelchangeCombo1()
{
int nIndex = Lista.GetCount();

if(running!=TRUE){
int Modello;
CString NameModello;

Modello = Lista.GetCurSel();
Lista.GetLBText(Modello, NameModello);

NumVer=Modello;
}
else
MessageBoxA(NULL,"Non cambiare modello durante esecuzione",
NULL, MB_OK | MB_ICONERROR);
}
```

Appendice C

Codice per versare liquido

```
.PROGRAM ruota()

        SET home = TRANS(0,0,-900,0,180,180)

        SPEED 10
        MOVE home

        xc = 20  ;differenza in x dei due punti
        zc = 200  ;differenza in z dei due punti

        d = SQRT(xc*xc+zc*zc) ;raggio circonferenza
        ang = 180+ATAN2(zc,xc) ;calcolo angolo compreso

        SPEED 10 ALWAYS

        CPON ALWAYS

        fi = 0
        N=20 ;valore in gradi da ruotare
        dx=50 ;valore per eventuale traslazione

        FOR i = 0 TO N ;il ciclo effettua N spostamenti
            fi = i
            ;impostazione di movimento sia avanti che indietro
            IF (i >= N/2) THEN
                fi = N-i
            END
```

```
            ;spostamento su coordinate definite rispetto
            ;alla posizione di home
            MOVE home:TRANS(dx,0,0):
            TRANS(xc+d*COS(ang+3*fi),0,zc+d*SIN(ang+3*fi),0,0,3*fi)

        END
.END

; NB:
; Tutti gli angoli devono essere espressi in gradi
; Viene aggiunta una matrice per eventuale traslazione
; di dx durante versamento
```